Combating Bufferbloat in Multi-Bottleneck Networks: Theory and Algorithms

Jiancheng Ye, *Member, IEEE*, Ka-Cheong Leung, *Senior Member, IEEE*, and Steven H. Low, *Fellow, IEEE, ACM*

***Abstract*— Bufferbloat is a phenomenon in computer networks where large router buffers are frequently filled up, resulting in high queueing delay and delay variation. More and more delay-sensitive applications on the Internet have made this phenomenon a pressing issue. Interacting with the Transmission Control Protocol (TCP), active queue management (AQM) algorithms run on routers play an important role in combating bufferbloat. However, AQM algorithms have not been widely deployed due to complicated manual parameter tuning. Moreover, they are often designed and analyzed based on network models with a single bottleneck link, rendering their performance and stability unclear in multi-bottleneck networks. In this paper, we propose a general framework to combat bufferbloat in multi-bottleneck networks. We first present an equilibrium analysis for a general multi-bottleneck TCP/AQM system and provide sufficient conditions for the uniqueness of an equilibrium point in the system. We then decompose the system into single-bottleneck subsystems and derive sufficient conditions for the local asymptotic stability of the subsystems. Using our framework, we develop an algorithm to compute the equilibrium point of the system. We further present a case study to analyze the stability of the recently proposed Controlled Delay (CoDel) in multi-bottleneck networks and devise Self-Tuning CoDel to improve the system stability. Extensive numerical and packet-level simulation results not only verify our theoretical studies but also show that our proposed Self-Tuning CoDel significantly stabilizes queueing delay in multi-bottleneck networks, thereby mitigating bufferbloat.**

***Index Terms*— Active queue management, bufferbloat, congestion control, equilibrium, stability.**

I. Introduction

A. Motivation

THE Internet is playing an increasingly important role in providing real-time communication and entertainment for people via network applications that have stringent requirements on end-to-end delay. In recent years, inexpensive memory has led to large buffers installed in Internet routers. Excessive buffering of packets results in high queueing delay and large delay variation, which is recently termed "bufferbloat" [2]. Larger buffers and more delay-sensitive applications on the Internet have made bufferbloat a pressing issue.

Active queue management (AQM) has been considered the key to combating bufferbloat, since it can avoid full buffer and high queueing delay by dropping or marking packets probabilistically before a buffer gets full. Random Early Detection (RED) [3] proposed in 1993 is the first AQM. Numerous AQM algorithms have been proposed since then, such as Proportional Integral (PI) controller [4], Random Exponential Marking (REM) [5], and Adaptive Virtual Queue (AVQ) [6]. Although research on AQM started almost three decades ago, none of the proposed AQM algorithms has been widely deployed because of complicated parameter tuning and sensitivity of algorithm performance to parameter settings. Recently, Controlled Delay (CoDel) [7] has been proposed as a modern AQM to address bufferbloat by controlling the queueing delay to a target value. It has been shown via experiments that CoDel performs well across various network settings with a simplified set of default parameters. More recently, Cisco researchers have proposed Proportional Integral controller Enhanced (PIE) [8], which is a lightweight AQM scheme that can effectively control the average queueing delay to a reference value.

However, the existing AQM algorithms are usually designed and analyzed based on network models with a single bottleneck link. Consequently, their performance and stability are unclear in general networks with multiple bottleneck links, such as the Internet. We consider a network link that is fully utilized as a bottleneck link in this paper. Stability is an important property for a dynamical system which concerns whether the system with initial state near an equilibrium will converge to the equilibrium. We find that the recently proposed CoDel may yield unstable queueing delay in multi-bottleneck networks with the default parameters that have been shown to produce stable queueing delay in various single-bottleneck network scenarios [1]. Thus, it is necessary to investigate the performance and stability of AQM in multi-bottleneck networks. To the best of our knowledge, there is very limited research on AQM in such networks. In [9], the instability of RED was observed in a network with two bottleneck links and the root cause for this unstable case was analyzed using control theory. In [10], a mathematical model for TCP/RED system with multiple bottleneck links was developed, and the stability of the system was then analyzed based on Lyapunov stability theory. However, [9] only provided a case study for the instability of RED in a two-bottleneck network, whereas [10] simply assumed that there existed a unique equilibrium point in a multi-bottleneck TCP/RED system. In fact, it is possible to have infinitely many equilibria in the system, as shown in Section VI-A.

Manuscript received April 25, 2020; revised December 23, 2020; accepted February 14, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Joo. This work was supported in part by the Research Grants Council, Hong Kong, China, under Grant 17204614. A preliminary version of this work was presented in IEEE INFOCOM 2018 [1]. *(Corresponding author: Ka-Cheong Leung.)*

Jiancheng Ye is with the Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong (e-mail: jcye@eee.hku.hk).

Ka-Cheong Leung is with the School of Computer Science and Technology, Harbin Institute of Technology at Shenzhen, Shenzhen 518055, China (e-mail: kcleung@ieee.org).

Steven H. Low is with the Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125 USA, and also with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: slow@caltech.edu).

Digital Object Identifier 10.1109/TNET.2021.3066505

On the other hand, the parameters of existing AQM algorithms are usually fixed. However, there is generally no fixed parameter setting that can guarantee an AQM algorithm to perform well in all possible network scenarios. Thus, it is highly desirable to automate parameter tuning for AQM based on network conditions. In fact, AQM with self-tuning capability has been recommended by the Internet Engineering Task Force (IETF) in RFC 7567 [11]. In order to automatically tune the parameters of an AQM, one needs to find fundamental relationships between the AQM parameters and network conditions. This requires a thorough understanding of the TCP/AQM congestion control. Congestion control is a distributed mechanism that aims to share network resources efficiently among competing flows. It consists of a source algorithm and a link algorithm [12]. On the current Internet, the source algorithm is primarily the Transmission Control Protocol (TCP). The link algorithm is either a simple drop-tail algorithm or an AQM algorithm employed by a router.

We realize that AQM is the key to addressing bufferbloat, but there is still a lack of AQM deployment due to complicated manual parameter tuning [2]. To facilitate the wide deployment of AQM, it is necessary to automate its parameter tuning which usually involves the stability analysis of the corresponding TCP/AQM system. Therefore, we propose a general framework to analyze the equilibrium and stability of general multi-bottleneck TCP/AQM systems. The theoretical results obtained from the analysis can then guide us to design distributed self-tuning AQM algorithms, which can be implemented in bottleneck routers, so as to practically combat bufferbloat.

B. Related Work

Over the last three decades, a variety of AQM algorithms have been proposed in the literature. They can be roughly categorized into AQM with static parameter settings and self-tuning AQM. We first review representative AQM with static settings. RED [3] as the first AQM drops or marks packets with a probability computed by comparing the average queue length to the predefined minimum and maximum thresholds. PI [4] computes the packet dropping probability based on the deviations of the current queue length from a reference value and a queue length history. REM [5] achieves high utilization and low loss and delay by stabilizing both the input rate around the link capacity and the queue length around a small target. AVQ [6] adjusts the queue size and the capacity of a virtual buffer as a function of the arrival rate. CoDel [7] drops a departing packet if the queueing delay exceeds a predefined target delay for a time period called *interval*.

In contrast to AQM with static settings, self-tuning AQM can dynamically adjust parameters based on network conditions. Adaptive RED [13] has been proposed to increase the robustness of the original RED by adapting the parameter of the maximum packet dropping probability. Self-Tuning PI and Self-Tuning RED have been proposed in [14], where their parameters are automatically tuned in response to the online estimations of link capacity and traffic load. In [15], the authors have proposed a self-tuning feedback controller named NPD-RED, which automatically selects values for the parameters from their determined ranges. Using control theory, [16] has proposed two self-tuning controllers for AQM routers and properly adapted the system to load changes. The recently proposed PIE [8] employs an internal auto-tuning mechanism that can adjust its control parameters based on the computed packet dropping probability, so that it can make a tradeoff between system stability and response time. On the other hand, stability analysis of a TCP/AQM system is fundamental to the self-tuning of AQM parameters since it usually derives some stability conditions for the system, which explicitly relate the AQM parameters with the network conditions. The stability of RED and PI has been analyzed in [4]. By applying the Nyquist stability criterion, proper parameters for RED and PI are determined to ensure system stability. A sufficient stability condition for the TCP/RED system with a single link and heterogeneous sources has been derived in [17]. Using control theory, [18] has derived explicit conditions under which the TCP/RED system is stable and discussed the stability region.

It should be noted that the existing AQM algorithms are usually devised and analyzed based on models with a single bottleneck link. There is very limited research on AQM in multi-bottleneck networks. The stability of RED in a network with two bottleneck links has been analyzed in [9]. In [10], the authors have further developed a mathematical model for the TCP/RED system with multiple bottleneck links and analyzed the stability of the system using Lyapunov stability theory.

Although this paper focuses on the AQM solution to bufferbloat, we are aware that the approach of reducing the buffer size itself is also worth further exploration. The series of [19], [20], and [21] on buffer sizing has provided some related technologies and theory.

C. Contributions

In this paper, we systematically address bufferbloat in general multi-bottleneck networks, from both theoretical and practical perspectives. We make the following contributions:

- We present an equilibrium analysis for a general multi-bottleneck TCP/AQM system and provide sufficient conditions for the uniqueness of an equilibrium in the system.
- We conduct stability study for the multi-bottleneck TCP/AQM system with feedback delays and derive sufficient conditions for the local asymptotic stability of the system. Furthermore, we decompose the system into single-bottleneck subsystems and derive sufficient conditions for the local asymptotic stability of the subsystems.
- Our equilibrium and stability studies constitute a general framework for analyzing the equilibrium and stability of loss-based TCP/AQM systems with multiple bottleneck links. Utilizing the framework, one can investigate the stability of AQM algorithms in multi-bottleneck networks, and design novel self-tuning AQM algorithms.
- Based on our framework, we develop an algorithm to compute the equilibrium point of the system. Moreover, we present a case study to analyze the stability of CoDel in multi-bottleneck networks. We further propose Self-Tuning CoDel, which dynamically adjusts the *interval* parameter of CoDel, so as to stabilize the queueing delays at all the bottleneck links, thereby mitigating bufferbloat.
- We perform extensive numerical and packet-level simulations, which verify our theoretical studies and show that Self-Tuning CoDel significantly stabilizes queueing delay in multi-bottleneck networks.

The rest of the paper is organized as follows. Section II presents a general multi-bottleneck TCP/AQM system model. In Section III, we investigate the existence and uniqueness of an equilibrium point in the system. Section IV presents a stability study for the system. In Section V, we develop an algorithm to compute the equilibrium point and propose the Self-Tuning CoDel algorithm based on our framework.

TABLE I
NOTATIONS

Notation	Meaning
N	Number of groups of TCP flows
L	Number of bottleneck links
$W_i(t)$	Window size of TCP flows in Group i at Time t
$N_i(t)$	Number of TCP flows in Group i at Time t
$R_i(t)$	Round-trip time of TCP flows in Group i at Time t
$q_l(t)$	Queue length at Link l at Time t
$p_l(t)$	Packet dropping probability at Link l at Time t
C_l	Capacity of Link l
β_l	Target queue length at Link l
K_l	Algorithm-specific parameter of an AQM employed by Link l
$L(i)$	Set of bottleneck links that Flow i traverses
$F(l)$	Set of groups of TCP flows that traverse Link l
M_l	Number of groups of TCP flows in $F(l)$, i.e., $\lvert F(l)\rvert$
$\mathbf{S}$	$N \times L$ routing matrix, where $S_{il} = 1$ if Flow i uses Link l and 0 otherwise
W_i^*	Equilibrium window size of TCP flows in Group i
q_l^*	Equilibrium queue length at Link l
p_l^*	Equilibrium packet dropping probability at Link l
x_i^*	Equilibrium rate of TCP flows in Group i
T_{p_i}	Fixed round-trip propagation delay of Flow i
I_0	The *interval* parameter of CoDel

Section VI presents our simulation results. Finally, we conclude the paper with future research directions in Section VII.

II. System Model

In this section, we present a general multi-bottleneck TCP/AQM system model. A list of frequently used notations is shown in Table I.

A. General Multi-Bottleneck TCP/AQM System Model

We consider a general TCP/AQM network with N groups of TCP flows and L bottleneck links. For simplicity, we do not include non-bottleneck links in the model since they generally do not affect the flows passing through them. Some flows may have the same set of parameters and traversing path, and we can categorize them into a group.

A well-known fluid model for TCP interacting with AQM routers was proposed using stochastic differential equations in [22]. The fluid model describes the additive increase/multiplicative decrease (AIMD) behavior of TCP Reno [23] and the queue dynamics at a router buffer. In [10], the fluid model was extended to the general AIMD algorithm interacting with RED and multiple bottleneck links. Applying the modeling technique similar to [10], we develop our multi-bottleneck TCP/AQM system model that uses loss-based TCP Reno as the source algorithm and employs a general packet dropping probability function to capture the dynamics of a class of AQM algorithms. The general packet dropping probability function $p(t)$ has the following form: $p(t) = \min\left\{\left(K[q(t)-\beta]\right)^+, 1\right\}$, where $(a)^+ = \max\{a, 0\}$, $p(t)$ is the packet dropping probability at Time t, $q(t)$ is the queue length at Time t, β is the target queue length, and $K > 0$ is a parameter determined by an AQM. The general form indicates that the packet dropping probability increases with the deviation of the queue length from the target value. In fact, the packet dropping probability functions of RED [10] and CoDel [24] are in this general form. Thus, our model can be applied to a class of AQM algorithms that determine the packet dropping probability based on the difference between the queue length and a target value, such as CoDel and RED. On the other hand, we use loss-based TCP Reno as the source algorithm in our model since it is a fundamental and representative TCP algorithm. We note that CUBIC [25] is another popular loss-based TCP algorithm, which employs a different congestion window function. A new fluid model for CUBIC has recently been proposed in [26]. We make use of some preliminary results from [26] to extend our work for supporting CUBIC (see Section V-C). Here, we would like to point out that it would be too complicated if multiple TCP variants and multiple AQM algorithms are taken into consideration in the model simultaneously. Therefore, we fix on a fundamental TCP variant as the source algorithm and mainly focus on the analysis of AQM algorithms in this paper.

Our general multi-bottleneck TCP/AQM system model that contains N groups of TCP flows and L bottleneck links is described by (1) and (2):

$$\dot{W}_i(t) = \frac{1}{R_i(t)} - \frac{W_i(t)W_i(t-R_i(t))}{2R_i(t-R_i(t))} \cdot \sum_{l\in L(i)} \Big(K_l\big[q_l(t-R_i(t)) - \beta_l\big]\Big)^+ \tag{1}$$

$$\dot{q}_l(t) = \begin{cases} \displaystyle\sum_{i\in F(l)} \frac{N_i(t)W_i(t)}{R_i(t)} - C_l, & q_l(t) > 0 \\ \left(\displaystyle\sum_{i\in F(l)} \frac{N_i(t)W_i(t)}{R_i(t)} - C_l\right)^+, & q_l(t) = 0 \end{cases} \tag{2}$$

where $L(i)$ denotes the set of bottleneck links that Flow i traverses, $F(l)$ is the set of groups of flows that traverse Link l, $i = 1, \ldots, N$, and $l = 1, \ldots, L$ (see Table I for other notations).

In (1), there are N differential equations, each of which describes the window size evolution of TCP flows in an individual group. In the subsequent analysis, we use Flow i to denote a flow in Group i. It should be noted that $W_i(t) \geq 0$, and that the summation term in (1) denotes the end-to-end packet dropping probability experienced by Flow i. Here, we assume that dropping probabilities at the bottleneck links are generally small, so that they can be added up to approximate the end-to-end dropping probability [12]. There are L differential equations in (2), each of which models the dynamics of queue length at a bottleneck link. Note that, since $q_l(t) \geq 0$, the rate of change of $q_l(t)$ cannot be negative when $q_l(t) = 0$.

B. Network Equilibrium

The equilibrium of the multi-bottleneck TCP/AQM system (1)–(2) is defined by $\dot{W}_i = 0$ and $\dot{q}_l = 0$ for $i = 1, \ldots, N$ and $l = 1, \ldots, L$. Assume that $N_i(t)$ and $R_i(t)$ are fixed on their equilibrium values when the system is in equilibrium. That is, $N_i(t) = N_i$ and $R_i(t) = R_i$ for $i = 1, \ldots, N$. The equilibrium point of the system is determined by the following equations:

$$(W_i^*)^2 \cdot \sum_{l\in L(i)} \Big(K_l(q_l^* - \beta_l)\Big)^+ = 2 \tag{3}$$

$$\left(\sum_{i\in F(l)} \frac{N_i W_i^*}{R_i} - C_l\right)^+ = 0 \tag{4}$$

where $(W_1^*, \ldots, W_N^*, q_1^*, \ldots, q_L^*)$ represents an equilibrium point, $i = 1, \ldots, N$, and $l = 1, \ldots, L$.

In (3), we have $W_i^* > 0$, which indicates that the equilibrium window size should be positive. Let $p_l^* = \Big(K_l(q_l^* - \beta_l)\Big)^+$ be the equilibrium packet dropping probability at Link l. Since we only consider bottleneck links in our model, we can assume $p_l^* > 0$ for all l and remove the notation $()^+$. We rewrite (3) as:

$$\sum_{l \in L(i)} p_l^* = \frac{2}{(W_i^*)^2} \tag{5}$$

where $i = 1, \ldots, N$.

From (4), we have $\sum_{i \in F(l)} \frac{N_i W_i^*}{R_i} = C_l$ or $\sum_{i \in F(l)} \frac{N_i W_i^*}{R_i} < C_l$. The latter case implies that Link l is not fully utilized ($p_l^* = 0$ in this case), and it can be eliminated since we only consider bottleneck links in the model. Thus, (4) can be simplified as:

$$\sum_{i \in F(l)} \frac{N_i W_i^*}{R_i} = C_l \tag{6}$$

where $l = 1, \ldots, L$.

Now, (5) and (6) represent the system in equilibrium, where $(W_1^*, \ldots, W_N^*, p_1^*, \ldots, p_L^*)$ is an equilibrium point. We consider p_l^* $(0 < p_l^* < 1)$ instead of q_l^* in the equilibrium study.

We define an $N \times L$ routing matrix $\mathbf{S}$, where entry $S_{il} = 1$ if Flow i uses Link l and $S_{il} = 0$ otherwise ($i = 1, \ldots, N$ and $l = 1, \ldots, L$). Furthermore, define the following vectors and matrices (we use bold fonts to denote vectors and matrices):

$$\begin{aligned}
\mathbf{W} &= [W_1^*, W_2^*, \ldots, W_N^*]^T, \\
\mathbf{E} &= \Big[\frac{2}{(W_1^*)^2}, \frac{2}{(W_2^*)^2}, \ldots, \frac{2}{(W_N^*)^2}\Big]^T, \\
\mathbf{P} &= [p_1^*, p_2^*, \ldots, p_L^*]^T, \\
\mathbf{C} &= [C_1, C_2, \ldots, C_L]^T, \quad \mathbf{A} = \mathrm{diag}\Big(\frac{N_i}{R_i}\Big).
\end{aligned}$$

Note that $\mathbf{E}$ is uniquely determined by $\mathbf{W}$. We introduce it in order to concisely express the equilibrium equations (5) and (6) in matrix form. $\mathbf{A}$ is an $N \times N$ diagonal matrix with entries $\frac{N_i}{R_i}$ $(i = 1, \ldots, N)$ in the main diagonal.

Then, (5) and (6) can be written in matrix form as follows:

$$\mathbf{SP} = \mathbf{E} \tag{7}$$

$$\mathbf{S^T A W} = \mathbf{C} \tag{8}$$

where $(\mathbf{W}, \mathbf{P})$ represents an equilibrium point of the system.

Here, (7) represents a system of N nonlinear equations in $\mathbf{W}$ and $\mathbf{P}$ (note that $\mathbf{E}$ is uniquely determined by $\mathbf{W}$), whereas (8) represents a system of L linear equations in $\mathbf{W}$ only.

III. Equilibrium Analysis

This section presents an analysis for the existence and uniqueness of an equilibrium point in the system (1)–(2). When the system is in equilibrium, an equilibrium point $(\mathbf{W}, \mathbf{P})$ is determined by (7) and (8). Thus, it is equivalent to study the existence and uniqueness of a solution of (7) and (8). Our equilibrium analysis is based on the following assumption.

Assumption 1: The $N \times L$ routing matrix $\mathbf{S}$ has full rank. That is, $\mathrm{rank}(\mathbf{S}) = \min\{N, L\}$.

The full rank property for a routing matrix is usually assumed in the literature, and it helps us seek conditions for the uniqueness of an equilibrium point. Properly performing routing configurations may help satisfy this assumption.

Since (8) is linear in $\mathbf{W}$ only, it is easy to study its solution first. In order to analyze the number of solutions for a system of linear equations, we use the Rouché-Capelli Theorem [27] in linear algebra.

First, we introduce a lemma, which affirms the existence of at least one equilibrium point in the system (1)–(2).

Lemma 1: Given N and L, there exists at least one equilibrium point in the system (1)–(2).

Proof: See Appendix A for a detailed proof. In addition, we notice that [28] presents a method to prove the existence of network equilibrium based on Nash equilibrium. It is also feasible to adopt that method to prove Lemma 1. ■

We then analyze the solutions of (7) and (8) based on the relationship between N and L, where $N \geq 1$ and $L \geq 1$.

A. Case I: $N < L$

In the case of $N < L$, we have $\mathrm{rank}(\mathbf{S}) = N$. By the definition of $\mathbf{A}$, $\mathrm{rank}(\mathbf{A}) = N$. Thus, we obtain $\mathrm{rank}(\mathbf{S^T A}) = N$. According to the Rouché-Capelli Theorem [27], (8) has solutions if and only if $\mathrm{rank}(\mathbf{S^T A}\big|\mathbf{C}) = \mathrm{rank}(\mathbf{S^T A}) = N$.

Note that the rank of the augmented matrix $(\mathbf{S^T A}\big|\mathbf{C})$ can either be $N + 1$ or N. In the former case, (8) does not have solutions, which implies that at least one link cannot be fully utilized. Consequently, there are less than L bottleneck links. In the latter case, (8) has a unique solution $\mathbf{W}$ since $\mathrm{rank}(\mathbf{S^T A})$ is equal to the number of unknowns.

The following theorem asserts the existence of infinitely many equilibrium points in the system when $N < L$.

Theorem 1: Suppose Assumption 1 holds. Given N and L such that $N < L$, there are infinitely many equilibrium points in the system (1)–(2).

Proof: The system has L bottleneck links that are fully utilized, which indicates that (8) is satisfied. That is, there exists a solution $\mathbf{W}$ that satisfies (8). Moreover, this solution $\mathbf{W}$ is unique since $\mathrm{rank}(\mathbf{S^T A}) = N$.

Next, we consider the solution of (7). Since $\mathbf{E}$ is uniquely determined by $\mathbf{W}$ and $\mathbf{W}$ is fixed on the unique solution of (8), (7) is now reduced to a system of N linear equations in $\mathbf{P}$ only. Then, (7) has solutions if and only if $\mathrm{rank}(\mathbf{S}) = \mathrm{rank}(\mathbf{S}\big|\mathbf{E})$. Note that $(\mathbf{S}\big|\mathbf{E})$ is an $N \times (L + 1)$ matrix and $N < (L + 1)$, so we have $\mathrm{rank}(\mathbf{S}\big|\mathbf{E}) = N = \mathrm{rank}(\mathbf{S})$. However, since $\mathrm{rank}(\mathbf{S})$ is less than L, which is the number of unknowns in (7), (7) has infinitely many solutions $\mathbf{P}$ according to the Rouché-Capelli Theorem. Thus, there are infinitely many equilibrium points $(\mathbf{W}, \mathbf{P})$ in the system when $N < L$. ■

From the proof of Theorem 1, we can see that $\mathbf{W}$ is unique, but there are infinitely many $\mathbf{P}$ in the system when $N < L$. See Section VI-A for such an example.

B. Case II: $N \geq L$

We now investigate the equilibrium of the system for the case of $N \geq L$. If $N = L$, $\mathbf{S}$ is a square matrix with full rank and hence has an inverse. By the definition of $\mathbf{A}$, $\mathbf{A}$ also has an inverse. Thus, we can directly solve for $\mathbf{W}$ in (8) as follows:

$$\mathbf{W} = \mathbf{A^{-1}}(\mathbf{S^{-1}})^{\mathbf{T}}\mathbf{C}. \tag{9}$$

Since $\mathbf{W}$ is uniquely determined by (9), $\mathbf{E}$ is then fixed. Thus, $\mathbf{P}$ can be uniquely solved from (7) as follows:

$$\mathbf{P} = \mathbf{S^{-1}E}. \tag{10}$$

The equilibrium point $(\mathbf{W}, \mathbf{P})$ is uniquely determined by (9) and (10) when $N = L$.

Next, we study the case of $N > L$. In this case, we find that $\text{rank}(\mathbf{S^T A}) = L < N$, which implies that there are infinitely many $\mathbf{W}$ that can satisfy (8). Hence, we cannot follow the approach used in the case of $N \leq L$ since $\mathbf{W}$ is not fixed.

We then use another method and study the standard form of the equilibrium equations, i.e., (5) and (6).

To prove the uniqueness of an equilibrium point in the system when $N > L$, we need an additional assumption, which is described as follows.

Assumption 2: The utility functions of TCP flows, denoted by $U_i(x_i)$, are strictly concave increasing in their rates x_i, and twice continuously differentiable in their domains.

The flow rate x_i is defined as $x_i = \frac{W_i}{R_i}$, where W_i and R_i are the window size and round-trip time of Flow i, respectively. This is a mild assumption. In fact, most proposed TCP algorithms, including TCP Reno, have strictly concave increasing utility functions according to [12].

In [29], the determination of the optimal flow rates of a network is formulated as a network utility maximization (NUM) problem. For the system (1)–(2), the corresponding NUM problem is defined as:

$$\max_{\mathbf{x}\geq 0} \sum_{i=1}^{N} N_i U_i(x_i) \tag{11}$$

$$\text{subject to } \sum_{i=1}^{N} S_{li} N_i x_i = C_l \tag{12}$$

where $\mathbf{x} = [x_1, \ldots, x_N]^T$ denotes the rate vector, N_i is the number of flows in Group i, S_{li} denotes entries of the $L \times N$ matrix $\mathbf{S^T}$, $i = 1, \ldots, N$, and $l = 1, \ldots, L$.

The equality constraint (12) corresponds to (6). The objective function (11) is a positive weighted sum of the utility functions and hence still strictly concave. Next, we prove the uniqueness of an equilibrium point by showing that the equilibrium point of the system (1)–(2) is the unique solution of (11)–(12).

Theorem 2: Suppose that Assumptions 1 and 2 hold. Given N and L such that $N \geq L$, there exists a unique equilibrium point in the system (1)–(2).

Proof: For $N = L$, we have shown that the equilibrium point is unique since it is uniquely determined by (9) and (10).

For $N > L$, the existence of an equilibrium point is proved by Lemma 1. Thus, we only need to prove the uniqueness of the equilibrium point in this case. Let $x_i^* = \frac{W_i^*}{R_i}$ be the equilibrium rate of Flow i.

Based on (5), we define a utility function $U_i(x_i)$ for Flow i such that its derivative equals the right-hand side of (5) [12]:

$$U_i'(x_i) = \frac{2}{R_i^2}\frac{1}{x_i^2}. \tag{13}$$

Thus, we have

$$U_i(x_i) = -\frac{2}{R_i^2}\frac{1}{x_i}. \tag{14}$$

This utility function is strictly concave increasing in $x_i > 0$.

Define the Lagrangian of (11)–(12) as:

$$L(\mathbf{x}, \mathbf{p}) = \sum_{i=1}^{N} N_i U_i(x_i) - \sum_{l=1}^{L} p_l \Big(\sum_{i=1}^{N} S_{li} N_i x_i - C_l \Big) \tag{15}$$

where $\mathbf{p} = [p_1, \ldots, p_L]^T$ denotes the vector of Lagrange multipliers associated with the equality constraint (12), $i = 1, \ldots, N$, and $l = 1, \ldots, L$.

A vector $\mathbf{x}^* = [x_1^*, \ldots, x_N^*]^T$ is a primal optimal solution of (11)–(12) if and only if there exists a vector $\mathbf{p}^* = [p_1^*, \ldots, p_L^*]^T$ such that the following Karush-Kuhn-Tucker (KKT) conditions are satisfied:

$$\mathbf{x}^* \geq 0, \quad \sum_{i=1}^{N} S_{li} N_i x_i^* = C_l \text{ for } l = 1, \ldots, L, \tag{16}$$

$$N_i U_i'(x_i^*) - N_i \sum_{l=1}^{L} S_{li} p_l^* = 0 \quad \text{for } i = 1, \ldots, N. \tag{17}$$

Note that complementary slackness and dual feasibility vanish from the KKT conditions since the problem does not have inequality constraints. Using (13), (17) can be rewritten as:

$$\frac{2}{R_i^2}\frac{1}{(x_i^*)^2} = \sum_{l=1}^{L} S_{li} p_l^* \quad \text{for } i = 1, \ldots, N. \tag{18}$$

We observe that (16) and (18) coincide with the equilibrium equations (5) and (6) (note that $x_i^* = \frac{W_i^*}{R_i}$). This means that an equilibrium point $(\mathbf{W}, \mathbf{P})$ of the system is an optimal solution of (11)–(12) and its dual problem, and vice versa.

Since $U_i(x_i)$ are strictly concave, the optimal solution $\mathbf{x}^*$ of (11)–(12) is unique. Thus, $\mathbf{W}$ is unique since $W_i^* = x_i^* R_i$. Next, we study (7). $\mathbf{E}$ is now uniquely fixed by $\mathbf{W}$. Since $\text{rank}(\mathbf{S}) = L$, the $L \times L$ matrix $\mathbf{S^T S}$ is invertible. From (7), we can solve for $\mathbf{P}$:

$$\mathbf{P} = (\mathbf{S^T S})^{-1}\mathbf{S^T E} \tag{19}$$

which is unique.

Thus, we have proved that there exists a unique equilibrium point $(\mathbf{W}, \mathbf{P})$ in the system when $N \geq L$. ■

IV. System Stability

In this section, we conduct a stability study for the system (1)–(2). Since the system is nonlinear and has time-varying delays, it is difficult to directly analyze its stability. Thus, we follow a standard approach used in [10], where a nonlinear TCP/RED system was linearized about the equilibrium point and Lyapunov stability theory was then adopted to study the linearized system. However, we notice that the matrix P in the Lyapunov function constructed in [10] may not always exist since the matrix $\overline{A}$ in their model cannot be proved to be always stable[1] [30]. A similar case also appears in our system (1)–(2). This hinders the subsequent algorithm design and thus leads us to consider simpler subsystems.

We name (1)–(2) the global system. In order to facilitate distributed algorithm design, we decompose the global system into single-bottleneck subsystems. For readability, here we mainly present the stability analysis of subsystems since it motivates our subsequent algorithm design. The stability study of the global system is presented in Appendix B.

[1]A square matrix is called stable matrix if each of its eigenvalues has negative real part.

When there exists a unique equilibrium point in the global system (1)–(2), we can formulate a single-bottleneck subsystem for every bottleneck Link l ($l = 1, \ldots, L$) as follows:

$$\dot{W}_i(t) = \frac{1}{R_i(t)} - \frac{W_i(t)W_i(t-R_i(t))}{2R_i(t-R_i(t))} \cdot \left\{ \sum_{\substack{j \in L(i) \\ j \neq l}} K_j(q_j^* - \beta_j) + \left(K_l \Big[q_l(t-R_i(t)) - \beta_l \Big] \right)^+ \right\}, \quad \text{for } i \in F(l) \tag{20}$$

$$\dot{q}_l(t) = \begin{cases} \sum_{i \in F(l)} \frac{N_i(t)W_i(t)}{R_i(t)} - C_l, & q_l(t) > 0 \\ \left(\sum_{i \in F(l)} \frac{N_i(t)W_i(t)}{R_i(t)} - C_l \right)^+, & q_l(t) = 0 \end{cases} \tag{21}$$

where q_j^* denotes the equilibrium queue length at Link j, and $W_i(t)$ for every $i \in F(l)$ and $q_l(t)$ for a fixed l are the variables in the subsystem of Link l.

Every subsystem (20)–(21) only considers the evolution of one bottleneck Link l and assumes that the queue lengths $q_j(t)$ at other links $j \neq l$ that Flow i ($i \in F(l)$) traverses are fixed on their unique equilibria q_j^*. This is a natural approximation for the global system. It is feasible because there is a unique equilibrium point in the global system, and q_j^* (or p_j^*) can be known to the subsystem, say, by Algorithm 1 in Section V-A.

Let $M_l = \big|F(l)\big|$. The subsystem (20)–(21) thus contains M_l groups of flows and one bottleneck link. We linearize (20)–(21) about the equilibrium point and obtain:

$$\delta\dot{W}_i(t) = -\frac{2}{W_i^* R_i}\delta W_i(t) - \frac{(W_i^*)^2}{2R_i} K_l \delta q_l(t - R_i) \tag{22}$$

$$\delta\dot{q}_l(t) = \sum_{i \in F(l)} \frac{N_i}{R_i}\delta W_i(t) - \frac{1}{C_l} \sum_{i \in F(l)} \frac{N_i W_i^*}{R_i^2} \delta q_l(t) \tag{23}$$

where $\delta W_i(t) = W_i(t) - W_i^*$ for $i \in F(l)$, $\delta q_l(t) = q_l(t) - q_l^*$.

In deriving (22)–(23), we assume that $N_i(t)$ and $R_i(t)$ are fixed on the equilibrium values, i.e., $N_i(t) = N_i$ and $R_i(t) = R_i$.

For every subsystem, we locally renumber the Flows $i \in F(l)$ so that $i = 1, \ldots, M_l$. Define the following matrices for every bottleneck Link l:

$$\mathbf{G}_l = \begin{bmatrix} \mathbf{G}_{l1} & \mathbf{0} \\ \mathbf{G}_{l2} & \mathbf{G}_{l3} \end{bmatrix} \tag{24}$$

where $\mathbf{G}_{l1} = \text{diag}\big(-\frac{2}{W_i^* R_i} \big)$ is an $M_l \times M_l$ diagonal matrix, $\mathbf{G}_{l2} = \big[\frac{N_1}{R_1}, \ldots, \frac{N_{M_l}}{R_{M_l}}\big]$ is a $1 \times M_l$ matrix, $\mathbf{G}_{l3} = -\frac{1}{C_l} \sum_{i=1}^{M_l} \frac{N_i W_i^*}{R_i^2}$, and $\mathbf{0}$ is a zero matrix.

$$\mathbf{H}_{li} = \begin{bmatrix} \mathbf{0} & \mathbf{H}_{li1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \text{for } i = 1, \ldots, M_l \tag{25}$$

where $\mathbf{H}_{li1} = \big[(H_{li1})_1, \ldots, (H_{li1})_{M_l}\big]^T$ with entries

$$(H_{li1})_j = \begin{cases} -\frac{(W_i^*)^2}{2R_i} K_l, & \text{if } j = i \\ 0, & \text{otherwise.} \end{cases}$$

Thus, $\mathbf{G}_l$ and $\mathbf{H}_{li}$ are $(M_l + 1) \times (M_l + 1)$ square matrices.

Define $\mathbf{z}_l(t) = [\delta W_1(t), \ldots, \delta W_{M_l}(t), \delta q_l(t)]^T$. The linearized subsystem (22)–(23) can be expressed in the following form:

$$\dot{\mathbf{z}}_l(t) = \mathbf{G}_l \mathbf{z}_l(t) + \sum_{i=1}^{M_l} \mathbf{H}_{li} \mathbf{z}_l(t - R_i). \tag{26}$$

We now show that all $\mathbf{G}_l$ are stable matrices.

Lemma 2: All $\mathbf{G}_l$ defined in (24) are stable matrices.

Proof: See Appendix C for a detailed proof. ∎

According to Lyapunov stability theory [30], since $\mathbf{G}_l$ defined in (24) is stable, there exists a unique positive definite matrix $\mathbf{P}_l$ that satisfies the following Lyapunov equation:

$$\mathbf{G}_l^{\mathbf{T}} \mathbf{P}_l + \mathbf{P}_l \mathbf{G}_l = -\mathbf{I} \tag{27}$$

where $\mathbf{I}$ is an $(M_l + 1) \times (M_l + 1)$ identity matrix, for every $l = 1, \ldots, L$.

We then study the stability of the subsystem by finding a Lyapunov function for it. We choose the following Lyapunov function candidate for every linearized subsystem (26):

$$V(\mathbf{z}_l) = \mathbf{z}_l^{\mathbf{T}} \mathbf{P}_l \mathbf{z}_l \tag{28}$$

where the positive definite matrix $\mathbf{P}_l$ is determined by (27).

The following theorem states the sufficient conditions for the local asymptotic stability of the subsystems.

Theorem 3: A single-bottleneck subsystem (20)–(21) is locally asymptotically stable if

$$0 < K_l < \frac{1}{2\mu J \|\mathbf{P}_l\| \sum_{i=1}^{M_l} \frac{(W_i^*)^2}{2R_i}} \tag{29}$$

where $J = \sqrt{\frac{\lambda_{max}(\mathbf{P}_l)}{\lambda_{min}(\mathbf{P}_l)}}$, $\lambda_{max}(\mathbf{P}_l)$ and $\lambda_{min}(\mathbf{P}_l)$ denote the largest and smallest eigenvalues of $\mathbf{P}_l$ defined in (27), respectively, $\|\mathbf{P}_l\|$ denotes the 2-norm of $\mathbf{P}_l$, and $\mu > 1$ is a constant.

Proof: We only need to show that $\dot{V}(\mathbf{z}_l) < 0$ if (29) is satisfied. Note that there are delayed terms $\mathbf{z}_l(t - R_i)$ in (26). We use the Razumikhin condition [31], which relates the past trajectory and the present state of the system, to bound the delayed terms.

Let $R_{max} = \max\{R_1, \ldots, R_{M_l}\}$. Whenever the following Razumikhin condition:

$$V(\mathbf{z}_l(\tau)) \leq \mu^2 V(\mathbf{z}_l(t)), \quad \text{for } t - R_{max} \leq \tau \leq t$$

holds for some constant $\mu > 1$ (note that $V(\mathbf{z}_l)$ defined in (28) is bounded), we have:

$$\lambda_{min}(\mathbf{P}_l)\|\mathbf{z}_l(\tau)\|^2 \leq \mathbf{z}_l^{\mathbf{T}}(\tau)\mathbf{P}_l\mathbf{z}_l(\tau) \leq \mu^2 \lambda_{max}(\mathbf{P}_l)\|\mathbf{z}_l(t)\|^2.$$

Thus, we can obtain

$$\|\mathbf{z}_l(\tau)\| \leq \mu J \|\mathbf{z}_l(t)\| \tag{30}$$

where $t - R_{max} \leq \tau \leq t$.

From the definition of $\mathbf{H}_{li}$ in (25), we have

$$\|\mathbf{H}_{li}\| = \frac{(W_i^*)^2}{2R_i} K_l. \tag{31}$$

From (26), (27), (28), (30), and (31), we can derive:

$$\dot{V}(\mathbf{z}_l) = \dot{\mathbf{z}}_l^{\mathbf{T}} \mathbf{P}_l \mathbf{z}_l + \mathbf{z}_l^{\mathbf{T}} \mathbf{P}_l \dot{\mathbf{z}}_l$$

$$= \mathbf{z}_l^{\mathbf{T}}(t)\Big(\mathbf{G}_l^{\mathbf{T}}\mathbf{P}_l+\mathbf{P}_l\mathbf{G}_l\Big)\mathbf{z}_l(t)+2\sum_{i=1}^{M_l}\mathbf{z}_l^{\mathbf{T}}(t-R_i)\mathbf{H}_{li}^{\mathbf{T}}\mathbf{P}_l\mathbf{z}_l(t)$$

$$\leq -\mathbf{z}_l^{\mathbf{T}}(t)\mathbf{I}\mathbf{z}_l(t)+2\sum_{i=1}^{M_l}\|\mathbf{z}_l^{\mathbf{T}}(t-R_i)\|\|\mathbf{H}_{li}^{\mathbf{T}}\|\|\mathbf{P}_l\|\|\mathbf{z}_l(t)\|$$

$$\leq -\|\mathbf{z}_l(t)\|^2+2\mu J\sum_{i=1}^{M_l}\|\mathbf{P}_l\|\|\mathbf{H}_{li}\|\|\mathbf{z}_l(t)\|^2$$

$$= -\Big(1-2\mu J\|\mathbf{P}_l\|\sum_{i=1}^{M_l}\frac{(W_i^*)^2}{2R_i}K_l\Big)\|\mathbf{z}_l(t)\|^2. \quad (32)$$

If (29) is satisfied, then $\dot{V}(\mathbf{z}_l) < 0$ when $\mathbf{z}_l(t) \neq 0$. That is, $V(\mathbf{z}_l)$ is a Lyapunov function for (26). Thus, the subsystem described by (20) and (21) is locally asymptotically stable. ■

Remark 1: Recall that K_l denotes an algorithm-specific parameter of an AQM employed by Link l. If the parameter value is set within the range specified by (29), the single-bottleneck subsystem is locally asymptotically stable.

V. Algorithms

Based on our theoretical studies, here we present an algorithm to compute the equilibrium point of the multi-bottleneck TCP/AQM system, and devise Self-Tuning CoDel algorithm to combat bufferbloat in multi-bottleneck networks.

A. Algorithm for Computing the Equilibrium Point

According to Theorem 2, there exists a unique equilibrium point in the system when $N \geq L$. Note that $N \geq L$ should be quite common in real networks. We now propose Algorithm 1 to compute the equilibrium point of the system, assuming that the routing matrix $\mathbf{S}$ is known to all AQM routers.

The essential idea of Algorithm 1 is that every AQM router needs to know all the packet dropping probabilities at the bottleneck links in order to compute the window sizes of the TCP flows from (5). In fact, Algorithm 1 is motivated by the widely used Open Shortest Path First (OSPF) routing protocol [32]. In OSPF, there is a designated router (DR) for every network segment, which is responsible for gathering and forwarding link state updates. Thus, we can let a specific AQM router serve as a centralized DR in the system. For example, an AQM router with the smallest ID is selected as DR. All AQM routers periodically send packet dropping probability updates to DR. When receiving updates, DR updates its list of packet dropping probabilities and sends the updated list to all AQM routers. In this way, every AQM router can compute the equilibrium point of the system based on (5). Note that the routing matrix $\mathbf{S}$ can be easily known if all AQM routers use the OSPF routing protocol since OSPF maintains the whole network topology.

The computation of the equilibrium point is important since this information is usually required when properly tuning the AQM parameters so as to stabilize the system. Note that the proposed Algorithm 1 is a general algorithm in the sense that it is not limited to a specific AQM algorithm employed by a bottleneck router. It provides a general method to compute the equilibrium point of the multi-bottleneck TCP/AQM system. In fact, Algorithm 1 is used by our proposed Self-Tuning CoDel algorithm when tuning the parameter, as illustrated in the next subsection. It is easy to see that the time complexity of Algorithm 1 is $O(ML)$, where L is the number of AQM routers and M is the number of updates occurred before the system reaches the equilibrium.

Algorithm 1 Computation of the Equilibrium Point

Assume: Full rank routing matrix $\mathbf{S}$ is known and $N \geq L$

1: Every AQM router at a bottleneck Link l $(l = 1, \ldots, L)$ measures its packet dropping probability p_l periodically.
2: When there is a change in p_l or update timer expires, the corresponding router sends an update of p_l to a centralized designated router (DR), which maintains a list of all p_l.
3: Upon receiving updates of p_l, DR updates its list and then sends the updated list to all AQM routers.
4: When the changes of all p_l are within a small δ, the system is assumed to be in equilibrium. With the list of all p_l and $\mathbf{S}$, every router can compute all W_i^* by $W_i^* = \sqrt{\frac{2}{\sum_{l\in L(i)} p_l}}$, and thus obtains the equilibrium point $(\mathbf{W}, \mathbf{P})$.

Next, we further give a discussion on the centralized property of Algorithm 1 and its possible decentralization:

- *Centrality for Simplicity:* Algorithm 1 assumes that every router can know the packet dropping probabilities at all the bottleneck links, which is realizable by using a centralized DR in a single autonomous system (AS) running OSPF protocol. Such a centrality is simple compared to a potential distributed algorithm that may require much more communication overheads among different routers.
- *Possible Decentralization:* For scalability (e.g., across multiple ASes), one can follow the OSPF's mechanism that uses an AS boundary router (ASBR) [32] to exchange routing information with routers belonging to other ASes. Specifically, we can let the DR router in an AS serve as the ASBR which exchanges packet dropping probability information with ASBRs belonging to other ASes. Every ASBR updates its corresponding list and sends the updated list to all routers in its own AS. Another method for decentralization is to apply dual decomposition [33] to the NUM problem (11)–(12). This involves an iterative primal algorithm at each TCP source and an iterative dual algorithm at each link (router). Due to space limitations, here we only briefly mention possible methods for decentralization.
- *Influence of Imprecise Estimation:* Due to scalability issues or delayed feedbacks, both the centralized and distributed versions of Algorithm 1 may use imprecise information about the packet dropping probabilities to compute the equilibrium point. Fortunately, the equilibrium window size depends on the end-to-end packet dropping probability in the order of $(\sum_{l\in L(i)} p_l)^{-1/2}$. In Section VI-F, we will show that the performance of our proposed Self-Tuning CoDel (Algorithm 2) is not affected significantly given a reasonable estimation error.

B. Self-Tuning CoDel Algorithm

The stability of CoDel in single-bottleneck networks has been analyzed in [24] and [34]. However, its stability in multi-bottleneck networks is still unknown. Thus, here we consider a multi-bottleneck TCP/CoDel system, where all the bottleneck links employ CoDel as the AQM algorithm.

In [24], an approximate packet dropping probability function of CoDel has been derived, and it can be expressed as:

$$p_l(t) \approx \min\left\{\left(\frac{q_l(t)-\beta_l}{4C_l I_0^2(\lambda_0-C_l)}\right)^+, 1\right\} \tag{33}$$

where I_0 denotes the *interval* parameter of CoDel, $\lambda_0 > C_l$ is the arrival rate at the time when the queue length first exceeds the target queue length β_l, and C_l is the link capacity.

From (33), we can let $K_l = \frac{1}{4C_l I_0^2(\lambda_0-C_l)}$ for CoDel. Substituting this into (29), we can derive:

$$I_0 > \sqrt{\frac{\mu J\|\mathbf{P}_l\| \sum_{i=1}^{M_l} \frac{(W_i^*)^2}{2R_i}}{2C_l(\lambda_0-C_l)}}. \tag{34}$$

If the value of I_0 is set such that (34) is satisfied, then every single-bottleneck TCP/CoDel subsystem is locally asymptotically stable according to Theorem 3.

Note that in order to trigger a packet drop, CoDel requires a minimum elapsed time since the queue length exceeds β_l, which is denoted by I_0. Thus, I_0 also affects the response time of CoDel. In practice, we can choose a value of I_0 that is slightly larger than the lower bound in (34), so that system stability is achieved without sacrificing the response time.

From (34), we observe that the range of I_0 is determined by several factors and will vary if these factors change over time. A static value of I_0 cannot always guarantee the stability of the TCP/CoDel system. Thus, we propose Self-Tuning CoDel to dynamically adjust I_0 at every bottleneck link based on network conditions, as shown in Algorithm 2.

When the update timer expires, Self-Tuning CoDel firstly estimates the related parameters. The equilibrium window size W_i^* can be estimated by Algorithm 1. Here, it is worth noting that this is the only part where Self-Tuning CoDel needs global information about the packet dropping probabilities at related bottleneck links, in order to accurately estimate W_i^* ($i \in F(l)$) for the flows traversing it. We then follow the methods proposed in the literature to estimate other network parameters locally. For example, efficient methods for estimating the number of TCP flows and link capacity were proposed in [14]. In [35], Karn's algorithm was proposed to accurately estimate the round-trip time. To solve the Lyapunov equation and compute the eigenvalues, we can call MATLAB to perform these computations. Alternatively, there are many related numerical methods proposed in the literature, such as [36] for solving Lyapunov equations, and [37] for computing eigenvalues. Finally, I_0 is updated based on (34). Self-Tuning CoDel can be run locally at every bottleneck link, so as to stabilize the queueing delays at all the bottleneck links in a distributed manner.

Regarding the time complexity of Self-Tuning CoDel, we do not explicitly derive it since it depends on specific algorithms used to solve the Lyapunov equation and compute the eigenvalues. Instead, we point out that after decomposing the global multi-bottleneck system into single-bottleneck subsystems, the sizes of $\mathbf{G}_l$ and $\mathbf{P}_l$ are much smaller compared to that of $\mathbf{G}_g$ and $\mathbf{P}_g$ in the global system (see Appendix B), which helps facilitate related computations. In fact, we observe via extensive simulations that with relatively small sizes (i.e., $(M_l+1)\times(M_l+1)$) of $\mathbf{G}_l$ and $\mathbf{P}_l$, the Lyapunov equation and the eigenvalues in Algorithm 2 can be solved very quickly using MATLAB.

Algorithm 2 Self-Tuning CoDel

Input: Full rank routing matrix $\mathbf{S}$

1: **while** update timer expires or condition changes **do**
2: Estimate W_i^*, N_i, R_i for every Flow $i \in F(l)$, and λ_0.
3: Construct $\mathbf{G}_l$ defined in (24).
4: Solve the Lyapunov equation (27) for $\mathbf{P}_l$ based on $\mathbf{G}_l$.
5: Compute $\lambda_{max}(\mathbf{P}_l)$ and $\lambda_{min}(\mathbf{P}_l)$, $J \leftarrow \sqrt{\frac{\lambda_{max}(\mathbf{P}_l)}{\lambda_{min}(\mathbf{P}_l)}}$.
6: Compute the 2-norm of $\mathbf{P}_l$: $\|\mathbf{P}_l\| \leftarrow \lambda_{max}(\mathbf{P}_l)$.
7: $I_0 \leftarrow \sqrt{\frac{\mu J\|\mathbf{P}_l\| \sum_{i=1}^{M_l} \frac{(W_i^*)^2}{2R_i}}{2C_l(\lambda_0-C_l)}} + \epsilon$, where $\mu > 1$ and $\epsilon > 0$.
8: CoDel is running with the updated I_0.
9: **end while**

Algorithm 3 Modified Self-Tuning CoDel for CUBIC

Assume: CUBIC sources can update their slow start thresholds based on the feedbacks from the AQM routers.

1: Initially, estimate W_i^* for every Flow $i \in F(l)$.
2: Send the estimated W_i^* to corresponding Flow i, and Flow i updates its slow start threshold based on this value.
3: Run the original Self-Tuning CoDel (Algorithm 2).

C. A Simple Modification of Self-Tuning CoDel for CUBIC

Recall that our system model and stability analysis use TCP Reno as the source algorithm. CUBIC [25] is another popular loss-based TCP algorithm, which exhibits a different behavior of congestion window evolution. Recently, a new fluid model for CUBIC has been proposed in [26]. We revisit [26] and find that the CUBIC's fluid model only considers a single-bottleneck scenario, and that instead of directly modeling the evolution of CUBIC's congestion window (*cwnd*), the model derives the evolutions of the size of the *cwnd* immediately before loss and the amount of time elapsed since last loss. Thus, we cannot directly incorporate the CUBIC's fluid model into our multi-bottleneck TCP/AQM system by using a new congestion window function for CUBIC. More thorough theoretical analysis for CUBIC interacting with AQM routers in multi-bottleneck networks still needs to be conducted, and hence we leave it as our future work.

However, [26] indeed provides some helpful insights for stabilizing a CUBIC system, which motivate us to develop a quick fix for Self-Tuning CoDel such that it can also work well with CUBIC. As the theoretical results in [26] suggest, one can specify the initial slow start threshold of CUBIC to be close to its equilibrium window size, so that the CUBIC system is more likely to settle into its stable state. Motivated by this, we make a simple modification for Self-Tuning CoDel interacting with CUBIC, which is shown in Algorithm 3.

Initially, a router running the Modified Self-Tuning CoDel (Algorithm 3) estimates the equilibrium window size W_i^* for every Flow $i \in F(l)$ that traverses it. This can be done by using Algorithm 1. Note that the equilibrium window sizes of the CUBIC flows can be estimated by $W_i^* = \sqrt[4]{\frac{cR_i^3}{b(\sum_{l\in L(i)} p_l)^3}}$, where $b = 0.2$ and $c = 0.4$ according to [26]. The Line 4 of Algorithm 1 needs to compute W_i^* for CUBIC flows using this equation instead. The router then sends the estimated W_i^* to the corresponding Flow i, and Flow i will update its slow start threshold based on W_i^*. If Flow i receives multiple W_i^*

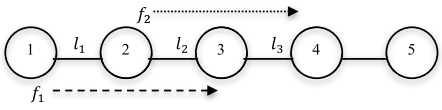


Fig. 1. Topology for Example 1, where there are infinitely many equilibria.

from different routers (since it traverses multiple bottleneck links), it will select the smallest one. After that, the router simply runs the original Self-Tuning CoDel.

Note that the Lines 1–2 of Algorithm 3 only need to be performed once during the initial stage. We also investigate the case where the source algorithm does not react to the feedback of W_i^* from the router (i.e., the case of CUBIC working with the original Self-Tuning CoDel), which is presented in Section VI-G.

VI. Performance Evaluation

This section presents simulation results obtained from ns-2 [38] and MATLAB, which evaluate our theoretical studies and proposed algorithms. We use ns-2 for packet-level simulations because CoDel [7] and PIE [8] are also evaluated by it.

A. Numerical Study for the Equilibrium of Multi-Bottleneck TCP/AQM System

First, we conduct numerical study for the equilibrium of the system by MATLAB. We use Example 1 to show that there can be infinitely many equilibrium points in the system, thereby verifying Theorem 1. The topology for Example 1 is shown in Fig. 1, where nodes denote routers. There are two groups of TCP flows ($N = 2$), i.e., f_1 and f_2. Each group contains ten flows, i.e., $N_1 = N_2 = 10$. f_1 traverses from Router 1 to Router 3 via Router 2, whereas f_2 goes from Router 2 to Router 4 via Router 3. There are three bottleneck links ($L = 3$) labeled by l_1, l_2, and l_3. The link capacities are as follows: $C_1 = 1250$ packets/s, $C_3 = 2500$ packets/s, and $C_2 = C_1 + C_3 = 3750$ packets/s. Let the round-trip times be $R_1 = 0.08$ s and $R_2 = 0.05$ s. Then, the equilibrium equation (6) is specified as:

$$\begin{cases} \dfrac{10}{0.08}W_1^* = 1250 \\ \dfrac{10}{0.08}W_1^* + \dfrac{10}{0.05}W_2^* = 3750 \\ \dfrac{10}{0.05}W_2^* = 2500. \end{cases} \tag{35}$$

We can obtain $W_1^* = 10$ and $W_2^* = 12.5$, which are unique. Substituting W_1^* and W_2^* into (5), we have:

$$\begin{cases} p_1^* + p_2^* = 0.02 \\ p_2^* + p_3^* = 0.0128. \end{cases} \tag{36}$$

It is obvious that (36) has infinitely many solutions. Thus, Example 1 shows that there are infinitely many equilibrium points in the multi-bottleneck TCP/AQM system if $N < L$.

Next, we present Example 2, where there is a unique equilibrium point in the system, and hence verifies Theorem 2. The topology for Example 2 is given in Fig. 2. There are three groups of TCP flows ($N = 3$), i.e., f_1, f_2, and f_3. f_1 traverses Routers 1–5. f_2 goes from Router 6 to Router 7 via Routers 2–3, whereas f_3 moves from Router 8 to Router 5 via Router 4. Let $N_1 = 50$, and $N_2 = N_3 = 5$. There are two bottleneck links ($L = 2$) labeled by l_1 and l_2. Let the link capacities be $C_1 = C_2 = 2500$ packets/s. The round-trip times are as

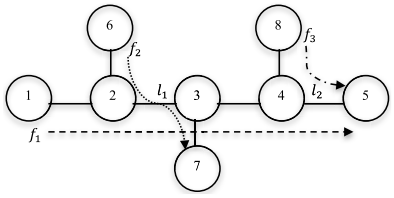


Fig. 2. Topology for Example 2, where there is a unique equilibrium.

follows: $R_1 = 0.12$ s, $R_2 = 0.07$ s, and $R_3 = 0.08$ s. Then, the equilibrium equations (5)–(6) are specified as:

$$\begin{cases} p_1^* + p_2^* = \dfrac{2}{(W_1^*)^2} \\ p_1^* = \dfrac{2}{(W_2^*)^2} \\ p_2^* = \dfrac{2}{(W_3^*)^2} \\ \dfrac{50}{0.12}W_1^* + \dfrac{5}{0.07}W_2^* = 2500 \\ \dfrac{50}{0.12}W_1^* + \dfrac{5}{0.08}W_3^* = 2500. \end{cases} \tag{37}$$

We can uniquely solve (37) and obtain: $W_1^* \approx 4.89$, $W_2^* \approx 6.49$, $W_3^* \approx 7.42$, $p_1^* \approx 0.047$, and $p_2^* \approx 0.036$. Hence, Example 2 shows that there is a unique equilibrium point in the multi-bottleneck TCP/AQM system if $N \geq L$.

We now further show the convergence of the system to the equilibrium point, and verify the feasibility of decomposing the global multi-bottleneck system into single-bottleneck subsystems. Here, the settings of Example 2 are used for illustration. In fact, the system (1)–(2) is a system of delay differential equations (DDEs) [39]. We can use the *dde23* solver [40] in MATLAB to solve DDEs with constant delays. Specifically, assuming $N_i(t) = N_i$ and $R_i(t) = R_i$ for $i = 1, \ldots, N$ in (1)–(2) and then using the settings of Example 2, we can solve the corresponding global system (1)–(2) using *dde23* and plot the solution evaluated in the interval $[0\ \text{s},\ 50\ \text{s}]$ in Fig. 3(a). In order to be consistent with the previous equilibrium analysis, we use the relationship $p(t) = \big(K[q(t) - \beta]\big)^+$ to replace the queue length by the packet dropping probability in (1)–(2). In addition, we set $K_1 = K_2 = \frac{1}{5000}$. Let the values of the TCP window sizes in the three groups be W_1, W_2, and W_3, respectively. The values of the packet dropping probabilities at the two bottleneck links are then denoted by p_1 and p_2, respectively. It can be observed in Fig. 3(a) that as time passes, the values of W_1, W_2, W_3, p_1, and p_2 converge to the equilibrium point $(4.89, 6.49, 7.42, 0.047, 0.036)$. Thus, we can see that the global system does converge to the unique equilibrium point computed previously in Example 2.

Based on the formulation of the subsystem given by (20)–(21), we can further solve the corresponding subsystems using the *dde23* solver. Specifically, for the subsystem at the bottleneck Link l_1, we fix the packet dropping probability at Link l_2 on $p_2 \approx 0.036$. Similarly, we fix the packet dropping probability at Link l_1 on the equilibrium value $p_1 \approx 0.047$ for the subsystem at Link l_2. Then, using the same settings of Example 2, we can solve the two corresponding subsystems using *dde23* and plot the solutions evaluated in the interval $[0\ \text{s},\ 50\ \text{s}]$, as shown in Fig. 3(b)–(c). It can be seen that the two subsystems converge to their unique equilibrium points as time passes, and that they actually converge faster than the global system. Furthermore, Fig. 3(a)–(c) demonstrate

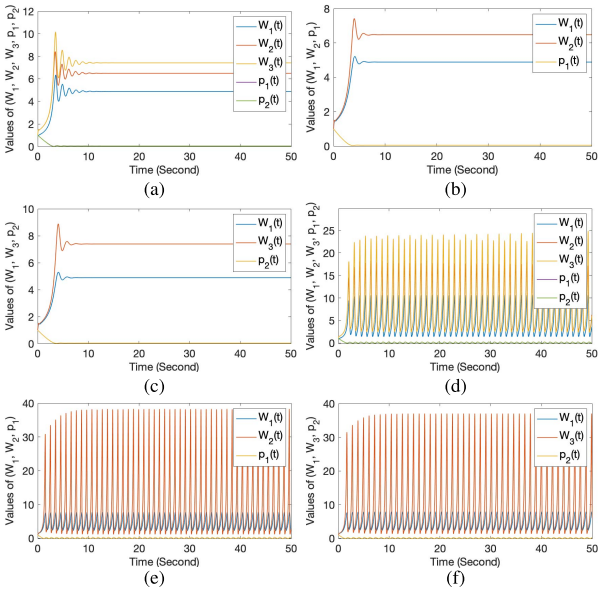


Fig. 3. Stable and unstable solutions to the DDEs describing the global multi-bottleneck system and the single-bottleneck subsystems. (a) Stable global system. (b) Stable subsystem at l_1. (c) Stable subsystem at l_2. (d) Unstable global system. (e) Unstable subsystem at l_1. (f) Unstable subsystem at l_2.

the feasibility of system decomposition in the sense that the equilibrium points of the two single-bottleneck subsystems and the equilibrium point of the global multi-bottleneck system coincide with each other.

B. Numerical Study for the Stability of Multi-Bottleneck TCP/AQM System

In this subsection, we show numerical results obtained from MATLAB to illustrate the stability and instability of the multi-bottleneck TCP/AQM system. In fact, Fig. 3(a)–(c) have demonstrated the stability of a multi-bottleneck TCP/AQM system based on the settings of Example 2, where the values of the two parameters K_1 and K_2 are set to $\frac{1}{5000}$. If we increase the values of K_1 and K_2, the multi-bottleneck TCP/AQM system will become unstable, as shown in Fig. 3(d). Recall that Theorem 3 and Theorem 4 (in Appendix B) only give the sufficient conditions for the local asymptotic stability of the subsystems and the global system, respectively. Thus, it may not be straightforward to determine the values of K_1 and K_2 that will lead to the instability of the system. However, we observe a trend that sufficiently large values of K_1 and K_2 will eventually induce the global system and the subsystems to become unstable. For example, Fig. 3(d) depicts an unstable global system when $K_1 = K_2 = \frac{1}{2700}$. If we further increase the values of K_1 and K_2 such that $K_1 = \frac{1}{1600}$ and $K_2 = \frac{1}{1800}$, both subsystems become unstable, as illustrated in Fig. 3(e)–(f). Therefore, the MATLAB numerical results shown in Fig. 3 indicate that in order to make the system and subsystems stable, the value of K_l should not be set too large.

C. Packet-Level Simulation Settings

We now conduct packet-level simulations using ns-2 to evaluate our proposed Self-Tuning CoDel. First, we describe our simulation settings as follows. Two representative loss-based TCP algorithms, namely NewReno [41] and CUBIC [25], are adopted as the source algorithms. NewReno is an enhanced

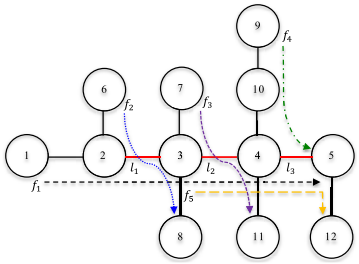


Fig. 4. Packet-level simulation topology: a three-bottleneck network.

version of Reno which improves Reno's fast recovery phase and still uses the same congestion window function described by (1). Thus, it fits our system model. NewReno is still the default TCP of some Windows and Linux operating systems according to [42] and [43]. CUBIC is more popular nowadays. We suggest a simple modification for Self-Tuning CoDel (Algorithm 3), which enables it to work well with CUBIC. Thus, we will present simulation results using NewReno and CUBIC in the next two subsections, respectively. Several representative AQM algorithms, including CoDel [7], PIE [8], and Adaptive RED [13], are used to compare with Self-Tuning CoDel. Our simulation topology is shown in Fig. 4, where nodes represent AQM routers. This is a complicated three-bottleneck network, where there are three bottleneck links (l_1, l_2, and l_3) and five groups of TCP flows (f_1, f_2, f_3, f_4, and f_5) traversing in different directions. Specifically, f_1 passes through Routers 1–5. f_2 goes from Router 6 to Router 8 via Routers 2–3, whereas f_3 moves from Router 7 to Router 11 via Routers 3–4. f_4 traverses Router 9, Router 10, Router 4, and then Router 5. f_5 transits Routers 3–5 and then Router 12. In our simulations, the long-lived TCP flows are generated by FTP applications. There are also short-lived HTTP flows and constant bit rate (CBR) flows going through Routers 1–5. Due to space limitations, here we only present simulation results using this three-bottleneck network topology. In fact, we have conducted extensive simulations and also provided simulation results using the two-bottleneck topology of Fig. 2 in our prior work [1]. Here, we use a more complicated topology to further evaluate the proposed Self-Tuning CoDel in terms of the stability of queueing delays at the AQM routers and the congestion window (*cwnd*) evolution of TCP flows. Note that the stability of queueing delays and the *cwnd* of TCP flows are two important metrics for TCP/AQM systems, since the former measures the performance of the AQM algorithms, whereas the latter reflects the throughput of the TCP flows.

The values of the main parameters used in our packet-level simulations are summarized in Table II. The simulation time is 600 seconds, and the TCP data packet size is 1000 bytes. The number of TCP flows varies over time. Specifically, from 0 s to 200 s, $N_1 = 200$, $N_2 = 40$, $N_3 = 30$, $N_4 = 20$, and $N_5 = 60$. From 200 s to 400 s, N_4 is increased to 40. From 400 s to 600 s, N_1 is then reduced to 140. Note that the equilibrium round-trip time of Flow i can be expressed as $R_i = T_{p_i} + \sum_{l \in L(i)} \frac{q_l^*}{C_l}$, where T_{p_i} denotes the fixed round-trip propagation delay of Flow i. To significantly differentiate the round-trip times of different flows, we set $T_{p_1} = 0.240$ s, $T_{p_2} = 0.130$ s, $T_{p_3} = 0.160$ s, $T_{p_4} = 0.100$ s, and $T_{p_5} = 0.210$ s. Let $C_1 = 60$ Mbps, $C_2 = 80$ Mbps, and $C_3 = 100$ Mbps when NewReno is used as the source algorithm. When the more advanced CUBIC is employed, we set the link capacities to larger values, as shown in Table II.

TABLE II
PACKET-LEVEL SIMULATION SETTINGS

Parameter	Value		
	[0 s, 200 s]	[200 s, 400 s]	[400 s, 600 s]
N_1	200	200	**140**
N_2	40	40	40
N_3	30	30	30
N_4	20	**40**	40
N_5	60	60	60
T_{p_1}	0.240 s		
T_{p_2}	0.130 s		
T_{p_3}	0.160 s		
T_{p_4}	0.100 s		
T_{p_5}	0.210 s		
C_1	60 Mbps (NewReno) or 80 Mbps (CUBIC)		
C_2	80 Mbps (NewReno) or 100 Mbps (CUBIC)		
C_3	100 Mbps (NewReno) or 120 Mbps (CUBIC)		

To make a fair comparison, we set the target delays of Self-Tuning CoDel, CoDel, PIE, and Adaptive RED to 20 ms at all the three bottleneck links. Other AQM parameters are set to default values. To make a good tradeoff between stability and response time, we limit the tuning of I_0 in Self-Tuning CoDel to the range $[100\text{ ms},\ 400\text{ ms}]$. If the computed lower bound in (34) is less than the default value 100 ms [7], I_0 can be set to 100 ms so as to align with CoDel. The update timer of Self-Tuning CoDel is set to 10 s. It can be reconfigured whenever needed.

D. Packet-Level Simulation Results Using NewReno

In this subsection, we present simulation results using NewReno as the source algorithm, which demonstrate that our proposed Self-Tuning CoDel can effectively improve the system stability and performance. We compare Self-Tuning CoDel with CoDel, PIE, and Adaptive RED in terms of the stability of queueing delays at the bottleneck links and the *cwnd* of TCP flows. The simulation topology and the values of the main parameters are shown in Fig. 4 and Table II, respectively. In addition, we use PackMime-HTTP [38] to simulate short-lived HTTP traffic. There are two new HTTP connections generated per second and ten CBR flows with the rate of 64 Kbps traversing Routers 1–5.

Fig. 5 plots the queueing delays yielded by the AQM algorithms at Link l_1, when NewReno is used. It can be seen in Fig. 5(b) that CoDel exhibits highly fluctuating queueing delay with some irregular spikes. We find that the default value of I_0 in CoDel does not satisfy (34) in this case, such that the stability of the subsystem at Link l_1 is not guaranteed. In contrast, Fig. 5(a) shows that our proposed Self-Tuning CoDel can consistently maintain stable queueing delay by properly adjusting I_0. PIE and Adaptive RED can also stabilize the queueing delay at Link l_1, as shown in Fig. 5(c)–(d).

Queueing delays at Link l_2 are presented in Fig. 6. It can be observed in Fig. 6(a)–(b) that CoDel also yields unstable queueing delay at Link l_2, whereas Self-Tuning CoDel still stabilizes the queueing delay at Link l_2. In addition, Adaptive RED produces frequently fluctuating queueing delay, as illustrated in Fig. 6(d). PIE still maintains stable queueing delay at Link l_2, which can be ascribed to its internal auto-tuning mechanism (PIE has auto-tuning for its control parameters so as to make a tradeoff between stability and response time).

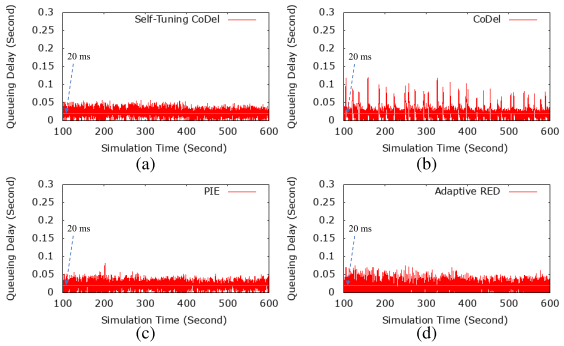


Fig. 5. Queueing delays at Link l_1, when NewReno is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

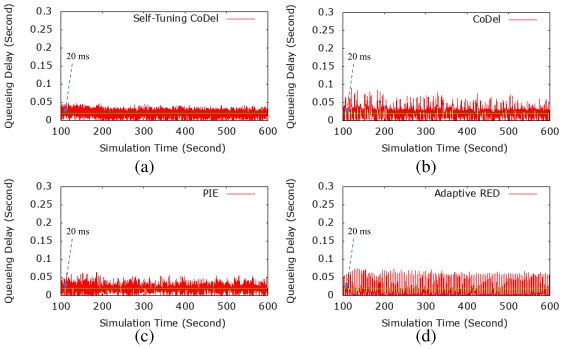


Fig. 6. Queueing delays at Link l_2, when NewReno is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

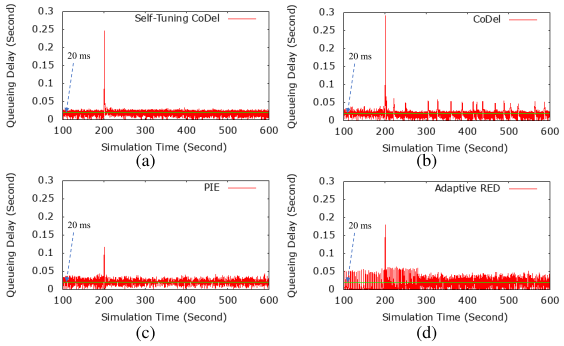


Fig. 7. Queueing delays at Link l_3, when NewReno is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

Fig. 7 depicts the queueing delays at Link l_3. Due to the arrivals of new f_4 flows, there is a sudden jump in the queueing delay around 200 s for the four AQM algorithms. Self-Tuning CoDel and PIE can adapt to the change in the number of TCP flows and stabilize the queueing delays, whereas CoDel exhibits irregular spikes in the queueing delay. Adaptive RED even shows unstable queueing delay in the beginning.

We now further present the *cwnd* of the TCP NewReno flows that are regulated by the AQM algorithms. Due to space limitations, here we only show the *cwnd* evolution of f_1, f_2, f_3, and f_4 flows (in fact, f_5 exhibits stable and similar evolution of *cwnd* for all the four AQM algorithms). Fig. 8 plots the evolution of the *cwnd* of a f_1 flow named Flow f_1 (note that all flows in the same group have similar characteristics for the *cwnd*). It can be seen in Fig. 8 that the four AQM algorithms yield similar evolution of *cwnd* for Flow f_1. Since Flow f_1 traverses all the three bottleneck links, it has a low value of *cwnd*. The *cwnd* evolution of a f_2 flow (i.e., Flow f_2) is presented in Fig. 9. The four AQM algorithms also induce similar evolution of *cwnd* for Flow f_2.

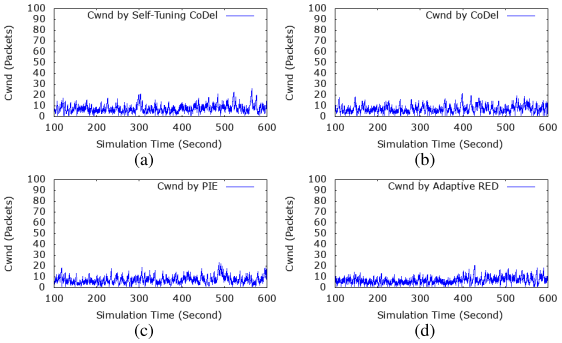


Fig. 8. Congestion window (*cwnd*) of NewReno Flow f_1. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

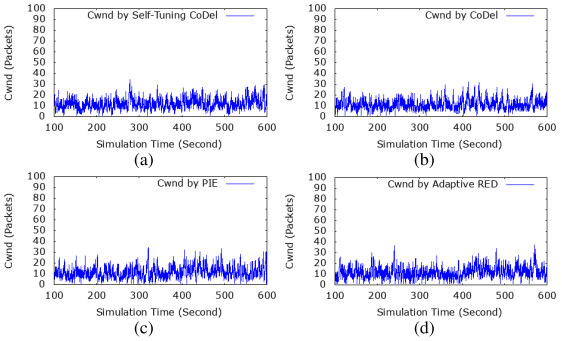


Fig. 9. Congestion window (*cwnd*) of NewReno Flow f_2. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

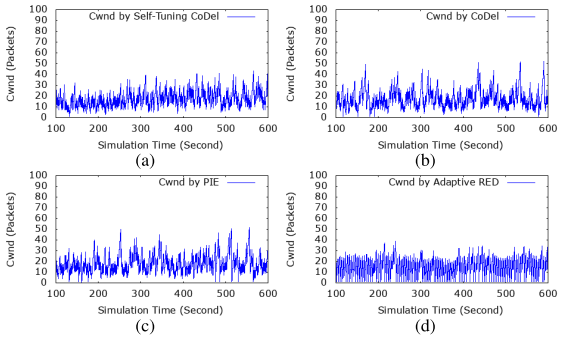


Fig. 10. Congestion window (*cwnd*) of NewReno Flow f_3. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

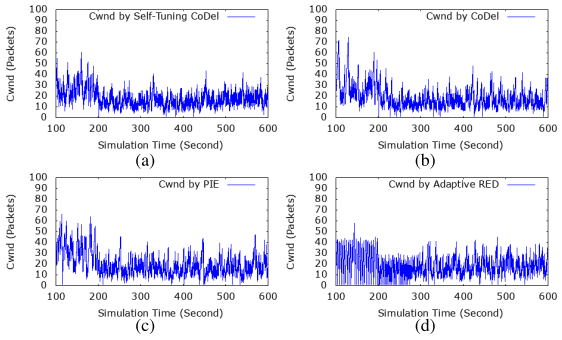


Fig. 11. Congestion window (*cwnd*) of NewReno Flow f_4. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

Fig. 10 plots the evolution of the *cwnd* of a f_3 flow (i.e., Flow f_3). It can be observed that Flow f_3 exhibits highly fluctuating *cwnd* when Adaptive RED is used (and it also frequently goes back to the slow start phase). In addition, CoDel and PIE produce significantly larger fluctuations in the *cwnd* of Flow f_3, compared to Self-Tuning CoDel. The *cwnd* evolution of a f_4 flow (i.e., Flow f_4) is shown in Fig. 11. Due

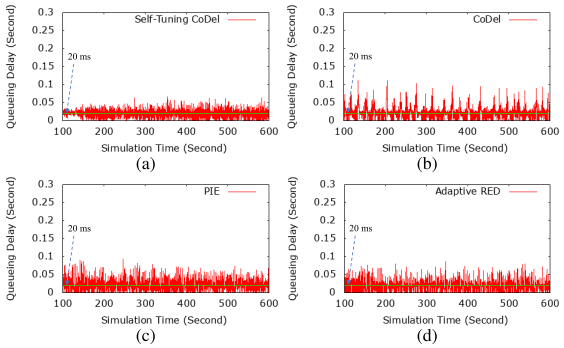


Fig. 12. Queueing delays at Link l_1, when CUBIC is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

to the arrivals of new flows, the *cwnd* of Flow f_4 is suddenly reduced around 200 s. As can be seen in Fig. 11(a), Self-Tuning CoDel can maintain the stability of the *cwnd* for Flow f_4. In contrast, CoDel, PIE, and Adaptive RED induce notably larger fluctuations in the *cwnd*.

From Fig. 5 to Fig. 11, we observe that Self-Tuning CoDel stabilizes the queueing delays at the bottleneck links without affecting the *cwnd* of the TCP flows. Indeed, the *cwnd* can also be stabilized.

E. Packet-Level Simulation Results Using CUBIC

This subsection further presents packet-level simulation results using CUBIC as the source algorithm. The simulation topology and the values of the parameters are almost the same as those used in the previous subsection except the following three changes. First, CUBIC is used as the source algorithm instead of NewReno. Second, to accommodate the more aggressive CUBIC flows, the capacity of each bottleneck link is increased by 20 Mbps such that $C_1 = 80$ Mbps, $C_2 = 100$ Mbps, and $C_3 = 120$ Mbps. Third, there are 10 new HTTP connections generated per second and 30 CBR flows with the rate of 64 Kbps traversing Routers 1–5 in the network. Note that a simple modification (Algorithm 3) has been incorporated into Self-Tuning CoDel in this subsection.

Fig. 12 presents the queueing delays yielded by the AQM algorithms at Link l_1, when CUBIC is used. It can be seen that Self-Tuning CoDel shows the smallest fluctuation in the queueing delay at Link l_1 among the four AQM algorithms. Moreover, Self-Tuning CoDel also stabilizes the queueing delay at Link l_2, whereas the other three algorithms (including PIE) exhibit highly fluctuating queueing delays at Link l_2, as illustrated in Fig. 13. Fig. 14 shows that the four algorithms are able to stabilize the queueing delay at Link l_3.

We also investigate the *cwnd* of the CUBIC flows that are controlled by the AQM algorithms. Fig. 15 plots the *cwnd* evolution of a f_1 flow (i.e., Flow f_1). Recall that since Flow f_1 traverses all the three bottleneck links, it maintains a low value of *cwnd*. The four AQM algorithms induce similar evolution of *cwnd* for Flow f_1. However, Self-Tuning CoDel yields a slightly higher *cwnd* for Flow f_1 and induces it to go back to the slow start phase less frequently, compared to PIE. The *cwnd* evolution of Flow f_2 is presented in Fig. 16. It can be seen that Self-Tuning CoDel maintains a stable evolution of *cwnd* for Flow f_2 over time. In contrast, CoDel and PIE show some high fluctuations in the *cwnd* of Flow f_2 and sometimes make the *cwnd* go back to the slow start phase. Fig. 17 presents the *cwnd* evolution of Flow f_3. We observe that Self-Tuning CoDel still yields a stable *cwnd* for Flow f_3, whereas CoDel,

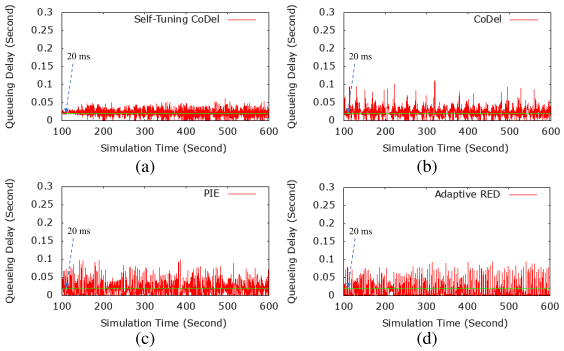


Fig. 13. Queueing delays at Link l_2, when CUBIC is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

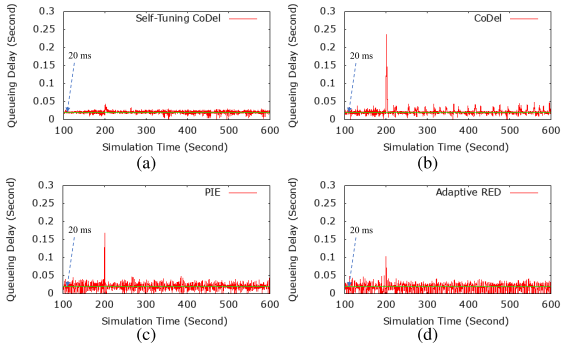


Fig. 14. Queueing delays at Link l_3, when CUBIC is the source algorithm. (a) Self-Tuning CoDel. (b) CoDel. (c) PIE. (d) Adaptive RED.

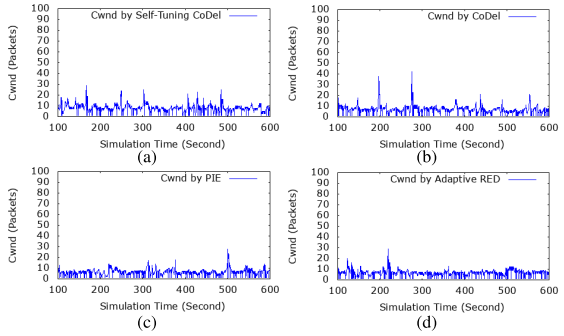


Fig. 15. Congestion window (*cwnd*) of CUBIC Flow f_1. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

PIE, and Adaptive RED exhibit highly fluctuating *cwnd* for Flow f_3 and also frequently force it to go back to the slow start phase. The *cwnd* evolution of Flow f_4 is shown in Fig. 18. Self-Tuning CoDel also maintains a more stable *cwnd* for Flow f_4, compared to the other three algorithms. The *cwnd* evolution of Flow f_5 is similar to that of Flow f_1. Due to space limitations, we omit it here.

The simulation results using CUBIC further demonstrate that Self-Tuning CoDel (with a simple modification) also works well with CUBIC and significantly outperforms PIE in terms of the stability of queueing delay and *cwnd*.

F. Influence of Imprecise Estimation of Packet Dropping Probability

In this subsection, we present one set of simulation results to show that Self-Tuning CoDel can tolerate reasonable estimation error in the computation of packet dropping probability, which corroborates the feasibility of Algorithms 1 and 2. In other words, although the centralized version of Algorithm 1 or a distributed version of it may use imprecise

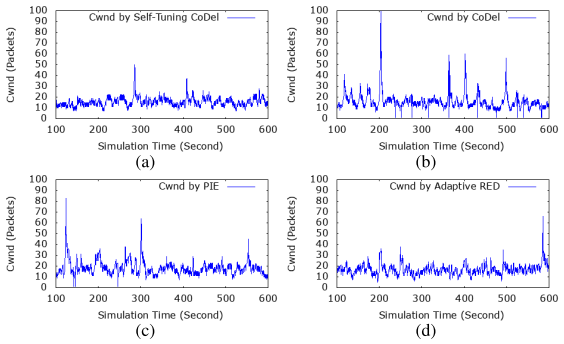


Fig. 16. Congestion window (*cwnd*) of CUBIC Flow f_2. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

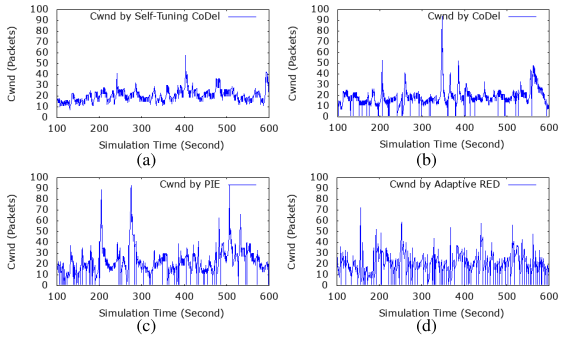


Fig. 17. Congestion window (*cwnd*) of CUBIC Flow f_3. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

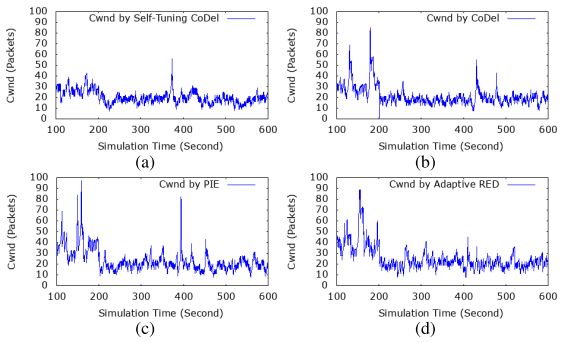


Fig. 18. Congestion window (*cwnd*) of CUBIC Flow f_4. (a) Using Self-Tuning CoDel. (b) Using CoDel. (c) Using PIE. (d) Using Adaptive RED.

information (due to scalability issues or delayed feedbacks) about the packet dropping probabilities at the bottleneck links to compute the equilibrium point, the performance of Self-Tuning CoDel is not affected significantly given a reasonable estimation error.

We use the same simulation settings mentioned in Section VI-D, except that we intentionally inject some "errors" into the estimation of packet dropping probabilities such that the computed end-to-end packet dropping probabilities (i.e., $\sum_{l \in L(i)} p_l$) for all the five groups of NewReno flows are underestimated by 20%. This estimation error case can correspond to the scenario where each AQM router at a Link l uses smaller end-to-end packet dropping probabilities (e.g., due to unavailable information from other links) to estimate the equilibrium window sizes W_i^* of the flows $i \in F(l)$ that traverse Link l based on (5) or (38).

Fig. 19 presents the queueing delay and *cwnd* produced by Self-Tuning CoDel for the above error case. It can be seen that the queueing delay at the bottleneck links and the *cwnd* of the flows can still be stabilized (Flow f_2 and Flow f_5 are not

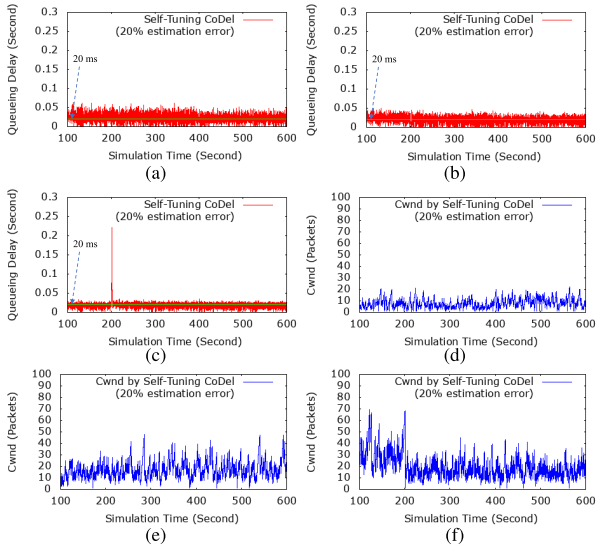


Fig. 19. Queueing delay and *cwnd* produced by Self-Tuning CoDel, when the end-to-end packet dropping probabilities for all the five groups of NewReno flows are underestimated by 20%. (a) Queueing delay at Link l_1. (b) Queueing delay at Link l_2. (c) Queueing delay at Link l_3. (d) *cwnd* of Flow f_1. (e) *cwnd* of Flow f_3. (f) *cwnd* of Flow f_4.

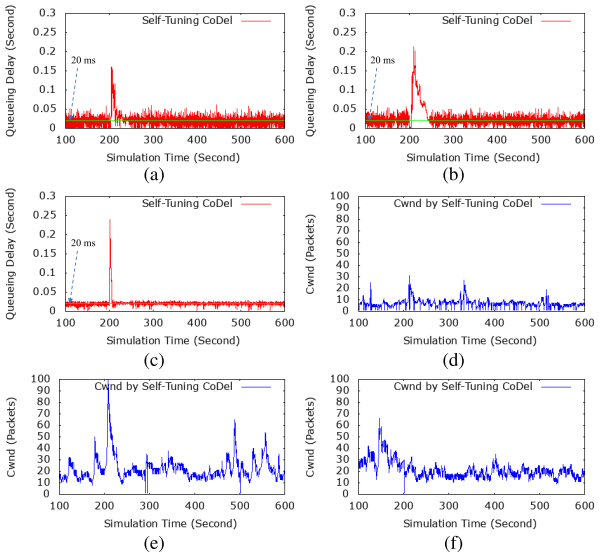


Fig. 20. Queueing delay and *cwnd* yielded by Self-Tuning CoDel, if the CUBIC sources do not update their initial slow start thresholds. (a) Queueing delay at Link l_1. (b) Queueing delay at Link l_2. (c) Queueing delay at Link l_3. (d) *cwnd* of Flow f_1. (e) *cwnd* of Flow f_3. (f) *cwnd* of Flow f_4.

shown due to space limitations). From the simulation results, we observe that a reasonable (e.g., 20%) underestimate of end-to-end packet dropping probabilities only yields a slightly larger computed lower bound of I_0, which does not affect the stability (but may have a minor effect on the response time). In fact, we have also tried the scenario of 20% overestimate of end-to-end packet dropping probabilities (which should be less common compared to the scenario of underestimate). The results confirm that Self-Tuning CoDel can also tolerate this reasonable overestimation error (they are not shown here due to space limitations).

G. CUBIC Interacting With the Original Self-Tuning CoDel

In this subsection, we further investigate the case where the CUBIC sources do not react to the feedbacks of W_i^* from the routers in Algorithm 3, which is equivalent to the case of CUBIC interacting with the original Self-Tuning CoDel.

We use the same simulation settings described in Section VI-E, except that the CUBIC sources do not update their slow start thresholds when receiving the feedbacks from the routers. Fig. 20 plots the queueing delay and *cwnd* yielded by Self-Tuning CoDel in this case. It can be observed that the arrivals of new f_4 flows around 200 s induce a sudden jump in the queueing delay at Link l_3 and this impact is then propagated to Link l_2 and Link l_1. After that, it takes much time to bring the queueing delay at Link l_2 back to the target value. We also observe that there are some occasional fluctuations in the *cwnd* of Flow f_3, especially when Flow f_3 seeks to increase *cwnd* right after 200 s. Therefore, the original Self-Tuning CoDel has performance degradation when working with CUBIC in multi-bottleneck networks. In fact, the performances of PIE and CoDel are also degraded significantly if CUBIC is used in this case, as shown in Section VI-E (note that the auto-tuning of PIE is still based on the TCP Reno's fluid model, which can be found in [8]).

To remedy this issue, we propose a quick fix for Self-Tuning CoDel (i.e., Algorithm 3) based on the theoretical results of [26]. The simulation results in Section VI-E have demonstrated the effectiveness of this quick fix. However, there is still a lack of solid stability analysis for CUBIC interacting with AQM routers in multi-bottleneck networks, and hence we consider filling this gap as one of our future research directions.

VII. Conclusion

In this paper, we have proposed a general framework for combating bufferbloat in multi-bottleneck networks. Specifically, we analyzed the equilibrium of a general multi-bottleneck TCP/AQM system and provided the sufficient conditions for the uniqueness of an equilibrium point in the system. We then conducted stability study for the multi-bottleneck system and the decomposed single-bottleneck subsystems by deriving the sufficient conditions for the local asymptotic stability of those systems. Based on the equilibrium and stability studies, we proposed an algorithm for computing the equilibrium point of the multi-bottleneck system. We further presented a case study to analyze the stability of the multi-bottleneck TCP/CoDel system and proposed Self-Tuning CoDel to improve the system stability and performance. By extensive numerical and packet-level simulations, we not only verified our theoretical studies but also demonstrated that Self-Tuning CoDel effectively stabilizes queueing delay and improves the *cwnd* of TCP flows in multi-bottleneck networks, compared to other representative AQM algorithms.

For future research, we plan to extend our framework by considering other popular TCP variants interacting with representative AQM algorithms in multi-bottleneck networks. Moreover, we plan to further study the interactions among different AQM algorithms employed by different bottleneck links in general networks containing wireless links.

Appendix A
Proof of Lemma 1

Proof: Recall that we have assumed $p_l^* > 0$ for all $l = 1, \ldots, L$ in the derivation of (5) and (6), since we only consider bottleneck links in the system.

From (5), we have:

$$W_i^* = \sqrt{\frac{2}{\sum_{l \in L(i)} p_l^*}} \tag{38}$$

where $i = 1, \ldots, N$. Given any $p_l^* > 0$ for all $l = 1, \ldots, L$, there exist W_i^* for all $i = 1, \ldots, N$, which are determined by (38).

Substituting (38) into (6) gives:

$$\sum_{i \in F(l)} \frac{N_i}{R_i} \sqrt{\frac{2}{\sum_{l \in L(i)} p_l^*}} = C_l \tag{39}$$

where $l = 1, \ldots, L$.

We claim that for the set of all feasible vectors $\mathbf{P}$ satisfying (38) and thus determining $\mathbf{W}$, there exists at least one $\mathbf{P}$ from this set, such that it also satisfies (39). To see why this claim holds, suppose that there is no such vector $\mathbf{P}$ from the above set, which can satisfy (39). That is, there is at least one link, say Link j, which always has $\sum_{i \in F(j)} \frac{N_i}{R_i} \sqrt{\frac{2}{\sum_{l \in L(i)} p_l^*}} < C_j$. This means that Link j cannot be fully utilized but with $p_j^* > 0$. It is impossible and contradicts the fact that all the L links are bottleneck links with $p_l^* > 0$ for all $l = 1, \ldots, L$. Note that, if Link j is not a bottleneck link, it can be eliminated with $p_j^* = 0$ without affecting the system.

Therefore, we can conclude that there exists at least one $\mathbf{P}$ that satisfies (38) and (39). This $\mathbf{P}$ determines $\mathbf{W}$ by (38) and hence this $(\mathbf{W}, \mathbf{P})$ is an equilibrium point of the system. ■

APPENDIX B
STABILITY OF THE GLOBAL MULTI-BOTTLENECK TCP/AQM SYSTEM

In this section, we present stability analysis of the global multi-bottleneck system (1)–(2). Similar to the analysis of subsystems, we first linearize (1)–(2) about the equilibrium point and obtain:

$$\delta \dot{W}_i(t) = -\frac{2}{W_i^* R_i} \delta W_i(t) - \sum_{l \in L(i)} \frac{(W_i^*)^2}{2R_i} K_l \delta q_l(t - R_i) \tag{40}$$

$$\delta \dot{q}_l(t) = \sum_{i \in F(l)} \frac{N_i}{R_i} \delta W_i(t) - \sum_{j=1}^{L} \left(\sum_{i \in F(l) \cap F(j)} \frac{1}{C_j} \frac{N_i W_i^*}{R_i^2} \delta q_j(t) \right) \tag{41}$$

where $i = 1, \ldots, N$ and $l = 1, \ldots, L$.

In (41), a term $\delta q_j(t)$ appears only when $F(l) \cap F(j) \neq \emptyset$, which means that there is at least one Flow i that traverses both links l and j.

Define the following matrices:

$$\mathbf{G}_g = \begin{bmatrix} \mathbf{G}_{g1} & \mathbf{0} \\ \mathbf{G}_{g2} & \mathbf{G}_{g3} \end{bmatrix} \tag{42}$$

where $\mathbf{G}_{g1} = \text{diag}\big(-\frac{2}{W_i^* R_i}\big)$ is an $N \times N$ diagonal matrix, $\mathbf{G}_{g2}$ is a $L \times N$ matrix with entries

$$(G_{g2})_{ij} = \begin{cases} \dfrac{N_j}{R_j}, & \text{if } j \in F(i) \\ 0, & \text{otherwise,} \end{cases}$$

$\mathbf{G}_{g3}$ is a $L \times L$ square matrix with entries

$$(G_{g3})_{ij} = \begin{cases} -\dfrac{1}{C_j} \displaystyle\sum_{k \in F(i)} \frac{N_k W_k^*}{R_k^2}, & \text{if } i = j \\ -\dfrac{1}{C_j} \displaystyle\sum_{k \in F(i) \cap F(j)} \frac{N_k W_k^*}{R_k^2}, & \text{otherwise,} \end{cases}$$

and $\mathbf{0}$ is a zero matrix.

$$\mathbf{H}_{gi} = \begin{bmatrix} \mathbf{0} & \mathbf{H}_{gi1} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \quad \text{for } i = 1, \ldots, N \tag{43}$$

where $\mathbf{H}_{gi1}$ is an $N \times L$ matrix with entries

$$(H_{gi1})_{jk} = \begin{cases} -\dfrac{(W_i^*)^2}{2R_i} K_k, & \text{if } j = i \text{ and } k \in L(i) \\ 0, & \text{otherwise.} \end{cases}$$

Thus, $\mathbf{G}_g$ and $\mathbf{H}_{gi}$ are $(N + L) \times (N + L)$ square matrices.

Define $\mathbf{z}(t) = [\delta W_1(t), \ldots, \delta W_N(t), \delta q_1(t), \ldots, \delta q_L(t)]^T$. The linearized system (40)–(41) can be expressed in the form:

$$\dot{\mathbf{z}}(t) = \mathbf{G}_g \mathbf{z}(t) + \sum_{i=1}^{N} \mathbf{H}_{gi} \mathbf{z}(t - R_i). \tag{44}$$

According to Lyapunov stability theory [30], there exists a positive definite matrix $\mathbf{P}_g$ that satisfies the following Lyapunov equation if and only if $\mathbf{G}_g$ defined in (42) is a stable matrix:

$$\mathbf{G}_g^{\mathbf{T}} \mathbf{P}_g + \mathbf{P}_g \mathbf{G}_g = -\mathbf{I} \tag{45}$$

where $\mathbf{I}$ is an $(N + L) \times (N + L)$ identity matrix.

Note that (42) is much more complicated than (24) when $L > 1$, so that it is unclear if $\mathbf{G}_g$ is always a stable matrix. Since we only seek sufficient conditions for system stability, we assume that there exists a positive definite matrix $\mathbf{P}_g$ that satisfies (45), so as to conduct the subsequent analysis.

We then choose the following Lyapunov function candidate for the linearized system (44):

$$V(\mathbf{z}) = \mathbf{z}^{\mathbf{T}} \mathbf{P}_g \mathbf{z} \tag{46}$$

where the positive definite matrix $\mathbf{P}_g$ is determined by (45).

Theorem 4: A multi-bottleneck TCP/AQM system described by (1) and (2) is locally asymptotically stable if there exists a positive definite matrix $\mathbf{P}_g$ satisfying (45) and

$$1 - 2\mu J_g \|\mathbf{P}_g\| \sum_{i=1}^{N} \|\mathbf{H}_{gi}\| > 0 \tag{47}$$

where $J_g = \sqrt{\frac{\lambda_{max}(\mathbf{P}_g)}{\lambda_{min}(\mathbf{P}_g)}}$, $\lambda_{max}(\mathbf{P}_g)$ and $\lambda_{min}(\mathbf{P}_g)$ denote the largest and smallest eigenvalues of $\mathbf{P}_g$, respectively, $\|\mathbf{P}_g\|$ denotes the 2-norm of $\mathbf{P}_g$, $\mathbf{H}_{gi}$ is defined in (43), and $\mu > 1$ is a constant.

Proof: Similar to the proof of Theorem 3, we use the Razumikhin condition [31] to bound the delayed terms and show that $\dot{V}(\mathbf{z}) < 0$ if (47) is satisfied.

Let $R_{max} = \max\{R_1, \ldots, R_N\}$. Whenever the following Razumikhin condition:

$$V(\mathbf{z}(\tau)) \leq \mu^2 V(\mathbf{z}(t)), \quad \text{for } t - R_{max} \leq \tau \leq t$$

holds for some constant $\mu > 1$, we can obtain

$$\|\mathbf{z}(\tau)\| \leq \mu J_g \|\mathbf{z}(t)\| \tag{48}$$

where $t - R_{max} \leq \tau \leq t$.

We then have:

$$\begin{aligned} &\dot{V}(\mathbf{z}) \\ &= \dot{\mathbf{z}}^{\mathbf{T}} \mathbf{P}_g \mathbf{z} + \mathbf{z}^{\mathbf{T}} \mathbf{P}_g \dot{\mathbf{z}} \\ &= \mathbf{z}^{\mathbf{T}}(t) \Big(\mathbf{G}_g^{\mathbf{T}} \mathbf{P}_g + \mathbf{P}_g \mathbf{G}_g \Big) \mathbf{z}(t) + 2 \sum_{i=1}^{N} \mathbf{z}^{\mathbf{T}}(t - R_i) \mathbf{H}_{gi}^{\mathbf{T}} \mathbf{P}_g \mathbf{z}(t) \end{aligned}$$

$$\leq -\mathbf{z}^{\mathbf{T}}(t)\mathbf{I}\mathbf{z}(t) + 2\sum_{i=1}^{N} \|\mathbf{z}^{\mathbf{T}}(t-R_i)\|\|\mathbf{H}_{gi}^{\mathbf{T}}\|\|\mathbf{P}_g\|\|\mathbf{z}(t)\|$$
$$\leq -\|\mathbf{z}(t)\|^2 + 2\mu J_g \sum_{i=1}^{N} \|\mathbf{P}_g\|\|\mathbf{H}_{gi}\|\|\mathbf{z}(t)\|^2$$
$$= -\Big(1 - 2\mu J_g\|\mathbf{P}_g\| \sum_{i=1}^{N} \|\mathbf{H}_{gi}\|\Big)\|\mathbf{z}(t)\|^2. \quad (49)$$

If (47) is satisfied, then $\dot{V}(\mathbf{z}) < 0$ when $\mathbf{z}(t) \neq 0$. Thus, $V(\mathbf{z})$ is a Lyapunov function for (44). Consequently, the multi-bottleneck system (1)–(2) is locally asymptotically stable. ■

Remark 2: In Theorem 4, the existence of a positive definite matrix $\mathbf{P}_g$ satisfying (45) becomes one of the conditions since $\mathbf{G}_g$ cannot be proved to be always stable. Besides, we can see that all the AQM parameters K_l are coupled in (47) via $\mathbf{H}_{gi}$, such that it is difficult to analyze them separately. These issues generally hinder the design of distributed algorithms.

APPENDIX C
PROOF OF LEMMA 2

Proof: Let λ be the eigenvalues of $\mathbf{G}_l$, and $\big|\mathbf{G}_l - \lambda\mathbf{I}\big| = 0$, where $\mathbf{I}$ is an $(M_l+1)\times(M_l+1)$ identity matrix. Expand $\big|\mathbf{G}_l - \lambda\mathbf{I}\big|$ along the last column using the Laplace expansion as follows:

$$\big|\mathbf{G}_l - \lambda\mathbf{I}\big| = \Big(-\frac{1}{C_l}\sum_{i=1}^{M_l}\frac{N_i W_i^*}{R_i^2} - \lambda\Big) \cdot \Big|\mathrm{diag}\Big(-\frac{2}{W_i^* R_i} - \lambda\Big)\Big| = 0.$$

Thus, we have $\lambda_i = -\frac{2}{W_i^* R_i}$ for $i = 1, \ldots, M_l$, and $\lambda_{M_l+1} = -\frac{1}{C_l}\sum_{i=1}^{M_l}\frac{N_i W_i^*}{R_i^2}$. Since all the eigenvalues of $\mathbf{G}_l$ have negative real parts, all $\mathbf{G}_l$ are stable matrices. ■

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the associate editor for their useful comments that have helped improve this article.

REFERENCES

[1] J. Ye, K.-C. Leung, V. O. K. Li, and S. H. Low, "Combating bufferbloat in multi-bottleneck networks: Equilibrium, stability, and algorithms," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 648–656.

[2] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the Internet," *Commun. ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012.

[3] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Netw.*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[4] C. V. Hollot, V. Misra, D. Towsley, and W. Gong, "Analysis and design of controllers for AQM routers supporting TCP flows," *IEEE Trans. Autom. Control*, vol. 47, no. 6, pp. 945–959, Jun. 2002.

[5] S. Athuraliya, S. H. Low, V. H. Li, and Q. Yin, "REM: Active queue management," *IEEE Netw.*, vol. 15, no. 3, pp. 48–53, May 2001.

[6] S. S. Kunniyur and R. Srikant, "An adaptive virtual queue (AVQ) algorithm for active queue management," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 286–299, Apr. 2004.

[7] K. Nichols and V. Jacobson, "Controlling queue delay," *ACM Queue*, vol. 10, no. 5, pp. 1–15, May 2012.

[8] R. Pan *et al.*, "PIE: A lightweight control scheme to address the bufferbloat problem," in *Proc. IEEE 14th Int. Conf. High Perform. Switching Routing (HPSR)*, Jul. 2013, pp. 148–155.

[9] D. Bauso, L. Giarre, and G. Neglia, "AQM stability in multiple bottleneck networks," in *Proc. IEEE Int. Conf. Commun.*, vol. 4, Jun. 2004, pp. 2267–2271.

[10] L. Wang, L. Cai, X. Liu, X. S. Shen, and J. Zhang, "Stability analysis of multiple-bottleneck networks," *Comput. Netw.*, vol. 53, no. 3, pp. 338–352, Feb. 2009.

[11] F. Baker and G. Fairhurst, *IETF Recommendations Regarding Active Queue Management*, document RFC 7567, IETF, Jul. 2015.

[12] S. H. Low, "A duality model of TCP and queue management algorithms," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 525–536, Aug. 2003.

[13] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," ACIRI, Tech. Rep., Aug. 2001.

[14] H. Zhang, D. Towsley, C. V. Hollot, and V. Misra, "A self-tuning structure for adaptation in TCP/AQM networks," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 302–303, Jun. 2003.

[15] N. Xiong *et al.*, "A novel self-tuning feedback controller for active queue management supporting TCP flows," *Inf. Sci.*, vol. 180, no. 11, pp. 2249–2263, Jun. 2010.

[16] Q. Chen and O. W. W. Yang, "On designing self-tuning controllers for AQM routers supporting TCP flows based on pole placement," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 10, pp. 1965–1974, Dec. 2004.

[17] S. H. Low, F. Paganini, J. Wang, S. Adlakha, and J. C. Doyle, "Dynamics of TCP/RED and a scalable control," in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies (INFOCOM)*, vol. 1, Jun. 2002, pp. 239–248.

[18] L. Tan, W. Zhang, G. Peng, and G. Chen, "Stability of TCP/RED systems in AQM routers," *IEEE Trans. Autom. Control*, vol. 51, no. 8, pp. 1393–1398, Aug. 2006.

[19] D. Wischik and N. McKeown, "Part I: Buffer sizes for core routers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 75–78, Jul. 2005.

[20] G. Raina, D. Towsley, and D. Wischik, "Part II: Control theory for buffer sizing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 79–82, Jul. 2005.

[21] M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, and T. Roughgarden, "Part III: Routers with very small buffers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 83–90, Jul. 2005.

[22] V. Misra, W.-B. Gong, and D. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 151–160, Oct. 2000.

[23] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[24] J. Ye and K.-C. Leung, "Adaptive and stable delay control for combating bufferbloat: Theory and algorithms," *IEEE Syst. J.*, vol. 14, no. 1, pp. 1285–1296, Mar. 2020.

[25] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[26] G. Vardoyan, C. V. Hollot, and D. Towsley, "Towards stability analysis of data transport mechanisms: A fluid model and an application," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 666–674.

[27] I. R. Shafarevich and A. O. Remizov, *Linear Algebra and Geometry*. Berlin, Germany: Springer-Verlag, 2013.

[28] A. Tang, J. Wang, S. H. Low, and M. Chiang, "Equilibrium of heterogeneous congestion control: Existence and uniqueness," *IEEE/ACM Trans. Netw.*, vol. 15, no. 4, pp. 824–837, Aug. 2007.

[29] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, Apr. 1998.

[30] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2002.

[31] K. Gu, V. L. Kharitonov, and J. Chen, *Stability of Time-Delay Systems*. Boston, MA, USA: Birkhäuser, 2003.

[32] J. Moy, *OSPF Version 2*, document IETF RFC 2328, Apr. 1998.

[33] M. Chiang, S. H. Low, A. R. Calderbank, and J. C. Doyle, "Layering as optimization decomposition: A mathematical theory of network architectures," *Proc. IEEE*, vol. 95, no. 1, pp. 255–312, Jan. 2007.

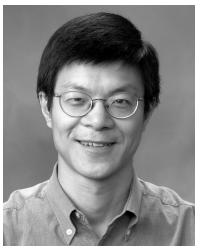
[34] C. Lai, S. H. Low, K.-C. Leung, and V. O. K. Li, "Pricing link by time," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 421–433, Jun. 2014.

[35] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 66–74, Jan. 1995.

[36] D. Kressner, "Block variants of Hammarling's method for solving Lyapunov equations," *ACM Trans. Math. Softw.*, vol. 34, no. 1, pp. 1–15, Jan. 2008.
[37] G. H. Golub and H. A. van der Vorst, "Eigenvalue computation in the 20th century," *J. Comput. Appl. Math.*, vol. 123, nos. 1–2, pp. 35–65, Nov. 2000.
[38] K. Fall and K. Varadhan, "The ns manual (formerly ns notes and documentation)," *The VINT Project*, Nov. 2011.
[39] R. D. Driver, *Ordinary and Delay Differential Equations*. New York, NY, USA: Springer-Verlag, 1977.
[40] L. F. Shampine and S. Thompson, "Solving DDEs in MATLAB," *Appl. Numer. Math.*, vol. 37, no. 4, pp. 441–458, Jun. 2001.
[41] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, *The NewReno Modification to TCP's Fast Recovery Algorithm*, document RFC 6582, IETF, Apr. 2012.
[42] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP congestion avoidance algorithm identification," *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1311–1324, Aug. 2014.
[43] The FreeBSD Project. *FreeBSD Manual Pages*. Accessed: 2021. [Online]. Available: https://www.freebsd.org/cgi/man.cgi?query=tcp

Jiancheng Ye (Member, IEEE) received the B.E. degree in network engineering from Sun Yat-sen University, Guangzhou, China, in 2008, the M.Phil. degree in computer science and engineering from The Hong Kong University of Science and Technology, Hong Kong, in 2011, and the Ph.D. degree in computer networking from The University of Hong Kong, Hong Kong, in 2018. He was a Software Engineer with Harmonic Inc., from 2011 to 2014, and a Post-Doctoral Fellow with the Department of Electrical and Electronic Engineering, The University of Hong Kong. His research interests include network congestion control, queue management, optimization and protocol design of computer networks, and machine learning.

Ka-Cheong Leung (Senior Member, IEEE) received the B.Eng. degree in computer science from The Hong Kong University of Science and Technology, Hong Kong, in 1994, and the M.Sc. degree in electrical engineering (computer networks) and the Ph.D. degree in computer engineering from the University of Southern California, Los Angeles, CA, USA, in 1997 and 2000, respectively. He was with the Nokia Research Center, Nokia Inc., Irving, TX, USA, from 2001 to 2002, Texas Tech University, Lubbock, TX, from 2002 to 2005, and The University of Hong Kong, Hong Kong, from 2005 to 2019. He is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include future Internet, smart city, vehicle-to-grid (V2G), machine learning, and wireless communications. He has co-guest edited a Special Issue of the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. He is a Subarea-Chair of the IEEE SIG on Intelligent Internet Edge (IIE). He is an Associate Editor of the IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE SYSTEMS JOURNAL, and *ACM/Springer Wireless Networks*.



Steven H. Low (Fellow, IEEE) received the B.S. degree in electrical engineering from Cornell and the Ph.D. degree in electrical engineering from Berkeley. He was with AT&T Bell Laboratories, Murray Hill, NJ, USA, and the University of Melbourne, Australia. He has held honorary/chaired professorship in Australia, China, and Taiwan. He is currently the F. J. Gilloon Professor with the Department of Computing and Mathematical Sciences and the Department of Electrical Engineering at Caltech. He is a Fellow of the ACM. He was a co-recipient of the IEEE Best Paper Awards in networking and power systems.